CSR5：跨平台稀疏矩阵 - 向量乘法的高效存储格式

汇报人：刘开放 邢光升
指导老师：龚春叶、甘新标、杨博

目录

稀疏矩阵向量乘法（SpMV）是许多应用中的基本算法。本文提出了一种新的稀疏矩阵存储格式CSR5（Compressed Sparse Row 5），它可在各种平台上执行高效的的SpMV，包括CPU，GPU（N卡和A卡）和Xeon Phi（Xeon Phi是由美国英特尔公司于北京时间2012年11月12日正式推出的首款60核处理器，Xeon Phi并非传统意义上的英特尔处理器(CPU)，它更像是与CPU协同工作的GPU(图形处理器)，其性能应该介于传统CPU和GPU之间）。

背　景

CSR格式及相关SpMV算法

CSR5格式及相关SpMV算法

测试结果及总结

背景

何为稀疏矩阵？对于这个概念估计很多同学已经有比较清楚的了解，为了方便后面论文的讲解，在这里再加啰嗦几句，其实稀疏矩阵通俗理解就是0值很多的矩阵。（矩阵中非零元素的个数远远小于矩阵元素的总数，并且非零元素的分布没有规律，通常认为矩阵中非零元素的总数比上矩阵所有元素总数的值小于等于0.05时，则称该矩阵为稀疏矩阵(sparse matrix))

$$G_{qr}=\begin{bmatrix} I & O & \cdots & O & G_{0,0} & G_{0,1} & \cdots & G_{0,r-1} \\ O & I & \cdots & O & G_{1,0} & G_{1,1} & \cdots & G_{1,r-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & G_{i,j} & \vdots \\ O & O & \cdots & I & G_{p-1,0} & G_{p-1,1} & \cdots & G_{p-1,r-1} \end{bmatrix}$$

CSR格式及相关SpMV算法

CSR格式

CSR格式由三个参数数据组成：

（1）row_ptr数组，它保存行的非零值的起始和结束指针。它的大小为m + 1，其中m是矩阵的行数

（2）col_idx数组，大小为nnz的存储非零值的列索引，其中nnz是矩阵中非零值的个数，

（3）val数组，大小为nnz的存储矩阵中的非零值。

$$A = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 3 \\ 0 & 1 & 0 & 2 \end{bmatrix}$$

row_ptr[] = [0 2 2 5 7]

col_idx[] = [0 2 0 2 3 1 3]

val[] = [1 2 1 2 3 1 2]

基于CSR格式的SpMV的并行算法

行块法

Algorithm 1 SpMV using the CSR-scalar method.

```
1: for i = 0 to m − 1 in parallel do
2:     y[i] ← 0
3:     for j = row_ptr[i] to row_ptr[i + 1] − 1 do
4:         y[i] ← y[i] + val[j] × x[col_idx[j]]
5:     end for
6: end for
```

定义：在给定的稀疏矩阵中，行彼此独立。可以在分解的行块上并行化SpMV操作。当物理处理单元的SIMD单元可用时，SIMD聚和操作可以获得更高的效率，分为标量法和向量法：

缺点：尽管具有良好的并行性，但在现代多处理器利用可伸缩性对于行块方法来说并非易事。主要因为矩阵的负载不平衡，矩阵由长度不均匀的行组成，如果矩阵的单行明显长于其他行，则可以仅使用单个核，而同一芯片中的其他核可以完全空闲。

分段和法

Algorithm 3 Segmented sum method CSR-based SpMV.

```
1:  MALLOC(*bit_flag, nnz)
2:  MEMSET(*bit_flag, FALSE)
3:  for i = 0 to m − 1 in parallel do                    ▷ Step 1
4:      bit_flag[row_ptr[i]] ← TRUE
5:  end for
6:  MALLOC(*product, nnz)
7:  for j = 0 to nnz − 1 in parallel do                  ▷ Step 2
8:      product[j] ← val[j] × x[col_idx[j]]
9:  end for
10: SEGMENTED_SUM(*product, *bit_flag)                   ▷ Step 3
11: for k = 0 to m − 1 in parallel do                    ▷ Step 4
12:     if row_ptr[k] = row_ptr[k + 1] then
13:         y[k] ← 0
14:     else
15:         y[k] ← product[row_ptr[k]]
16:     end if
17: end for
18: FREE(*bit_flag)
19: FREE(*product)
```

定义：分段和对阵列中每个段中的数据执行缩减和运算。在基于CSR的SpMV操作中，分段和方法将每个矩阵行视为一个段，并计算每行中生成的部分和。

使用分段和方法的SpMV操作由四个组成步骤：

（1）从row_ptr数组生成大小为nnz的辅助bit_flag数组。如果bit_flag中的条目的位置与第一个非零条目匹配，则将其标记为TRUE，否则标记为FALSE。

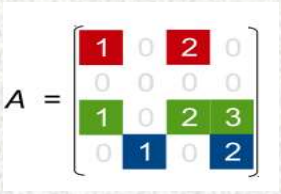
（2）计算所有中间结果到大小为nnz的数组

（3）执行并行分段求和并将结果生成数组

（4）将所有部分和收集到结果向量中

CSR格式及相关SpMV算法

结合图片说明：



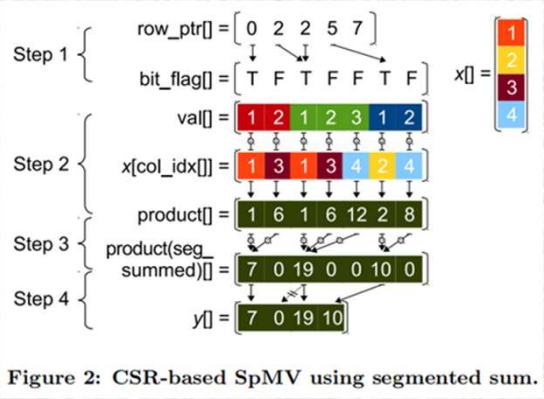
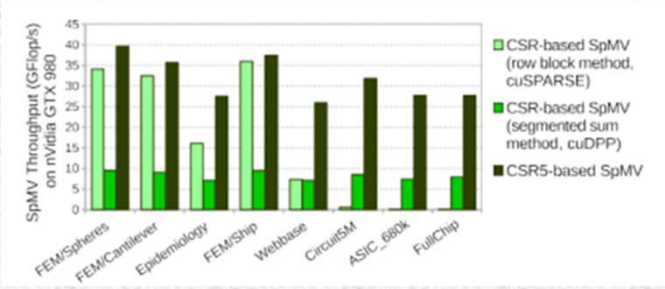


Figure 2: CSR-based SpMV using segmented sum.



文章针对不同数据集再N卡（980）上进行了测试，我们只看基于CSR的两种算法对比，会发现分段和法并不比行块法表现的突出，为什么会这样？我们可以看到步骤1是分散操作，步骤4是聚合操作

在这种情况下，更多的全局同步和全局访存操作可能会降低整体性能。研究发现，分段和法可能更适合基于COO（坐标存储格式）的SpMV。

CSR5格式及相关SpMV算法

CSR5

为了实现具有任何稀疏矩阵最佳负载平衡，文章首先将所有非零条目均匀地划分为多个相同大小的二维区块。因此，当执行并行SpMV操作时，计算核心可以消耗一个或多个二维区块，并且核心的每个SIMD通道可以处理区块的一列。整个CSR5格式的主要框架就是一组二维区块。

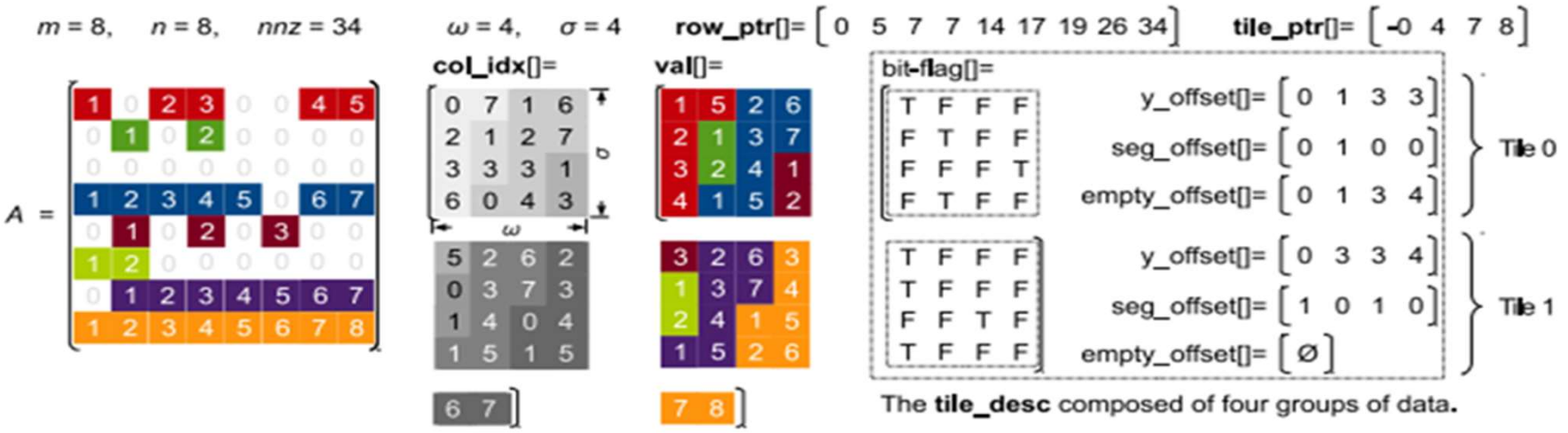


Figure 4: The CSR5 storage format of a sparse matrix A of size 8×8. The five groups of information include row_ptr, tile_ptr, col_idx, val and tile_desc.

CSR5格式及相关SpMV算法

CSR5

对照上面讲解的CSR格式，m，n是稀疏矩阵行数和列数，nnz是矩阵中非零元素个数。ω和σ是调整参数，其中ω是区块的宽度，σ是它的高度，CSR5格式只有这两个调整参数。此外，我们需要额外的信息来计算SpMV。对于每个区块，我们引入区块指针tile_ptr和区块描述符tile_desc。同时，直接集成经典CSR格式的三个数组，即行指针row_ptr，列索引col_idx和值val。

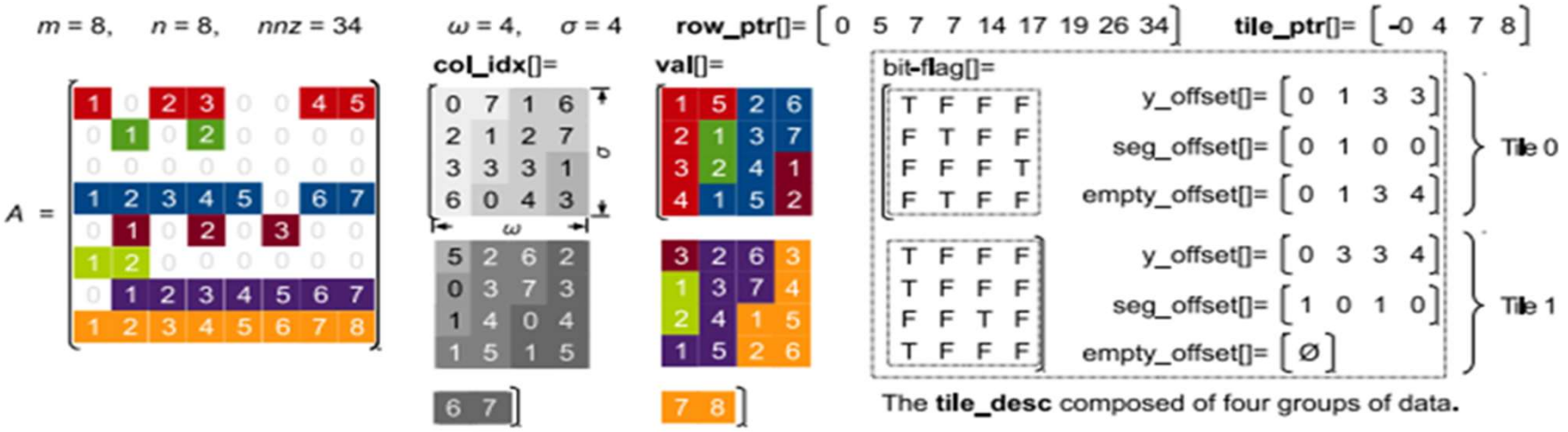


Figure 4: The CSR5 storage format of a sparse matrix A of size 8×8. The five groups of information include row_ptr, tile_ptr, col_idx, val and tile_desc.

CSR5格式及相关SpMV算法

CSR5

tile_ptr存储每个区块中的第一个元素的原始矩阵行索引，指示用于将其分段和存储到向量y的起始位置。空行与第一个非空的右邻居行具有完全相同的行指针信息，为了识别是否需要特定的过程，如果相应的区块对应的原始矩阵包含空行，我们在tile_ptr中将一个条目设置为负值。对于剩下未分配的尾巴，直接补齐到数组当中。

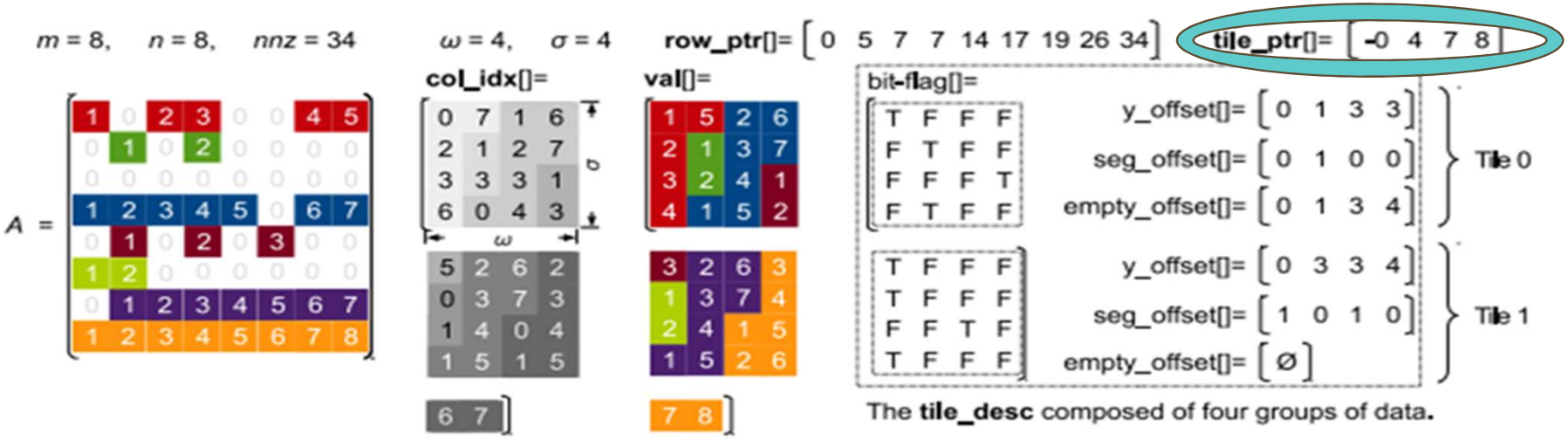


Figure 4: The CSR5 storage format of a sparse matrix A of size 8×8. The five groups of information include row_ptr, tile_ptr, col_idx, val and tile_desc.

CSR5格式及相关SpMV算法

CSR5

bit_flag非常简单，与原位转置的 col_idx 和 val 匹配。此外，每个图块的bit_flag的第一个条目设置为TRUE。其余位置1值设为TRUE，其他值设为FALSE。

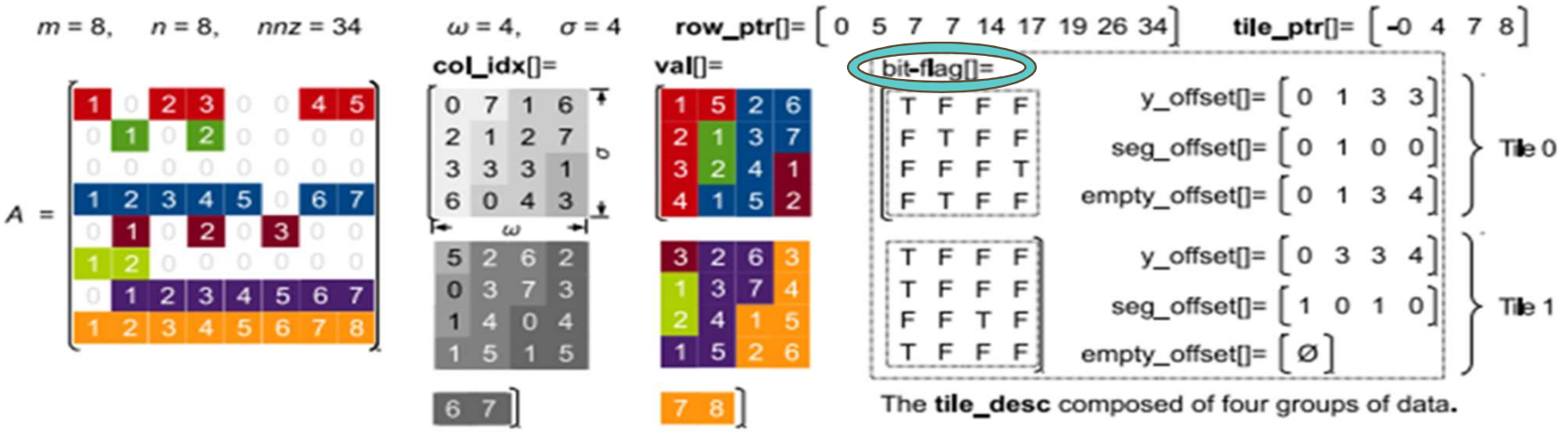


Figure 4: The CSR5 storage format of a sparse matrix A of size 8×8. The five groups of information include row_ptr, tile_ptr, col_idx, val and tile_desc.

CSR5格式及相关SpMV算法

CSR5

y_offset用于帮助每个区块中的列知道它们的分段和和存储到y的起始点。生成y_offset很简单：每列计算其先前列的bit_flag数组中的TRUE数。

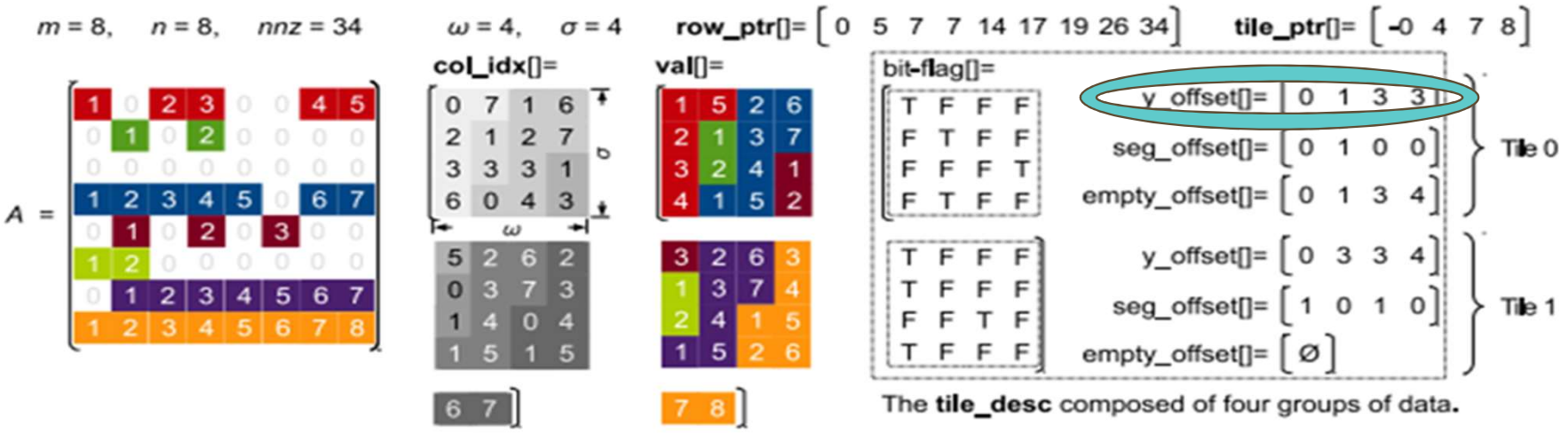


Figure 4: The CSR5 storage format of a sparse matrix A of size 8×8. The five groups of information include row_ptr, tile_ptr, col_idx, val and tile_desc.

CSR5格式及相关SpMV算法

CSR5

seg_offset为辅助信息数组，以便通过前缀和扫描来实现分段和，生成seg_offset也很简单，让每列搜索其右邻居列是否没有任何TRUE，如说是则该列标志置1，否则为0。

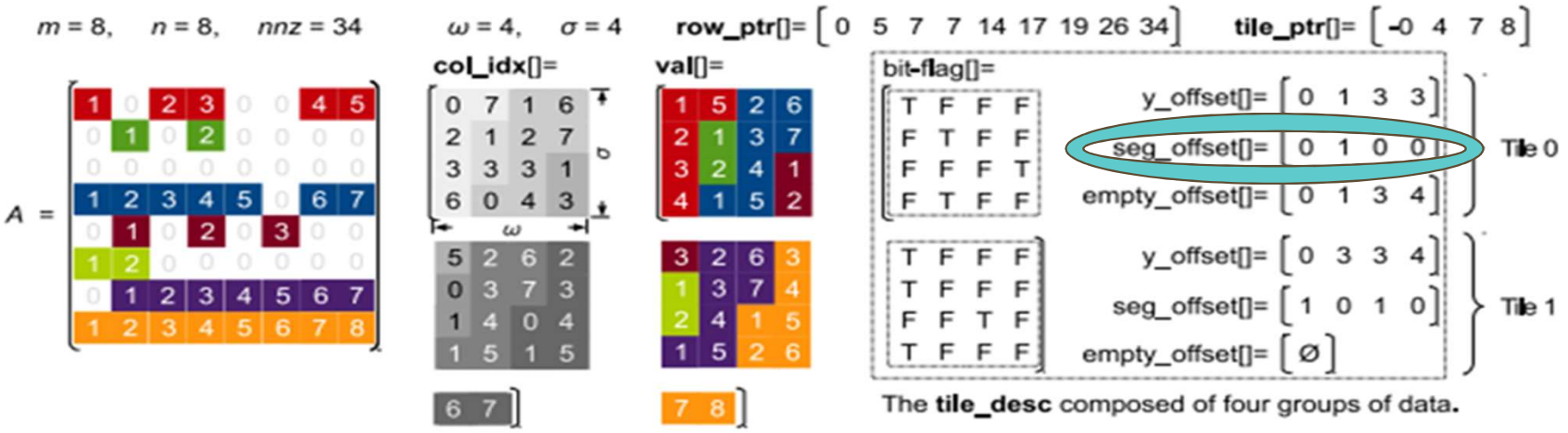


Figure 4: The CSR5 storage format of a sparse matrix A of size 8×8. The five groups of information include row_ptr, tile_ptr, col_idx, val and tile_desc.

CSR5格式及相关SpMV算法

CSR5

empty_offset只有区块对应原始矩阵中包括任何空行才生成，否则值为空。因为矩阵的空行同其临近行具有相同的行指针，y_offset将为其记录不正确的偏移量。为纠正这个问题设置此参数。因此，empty_offset的长度是块中的段的数量，即bit_flag中的TRUE的总数。

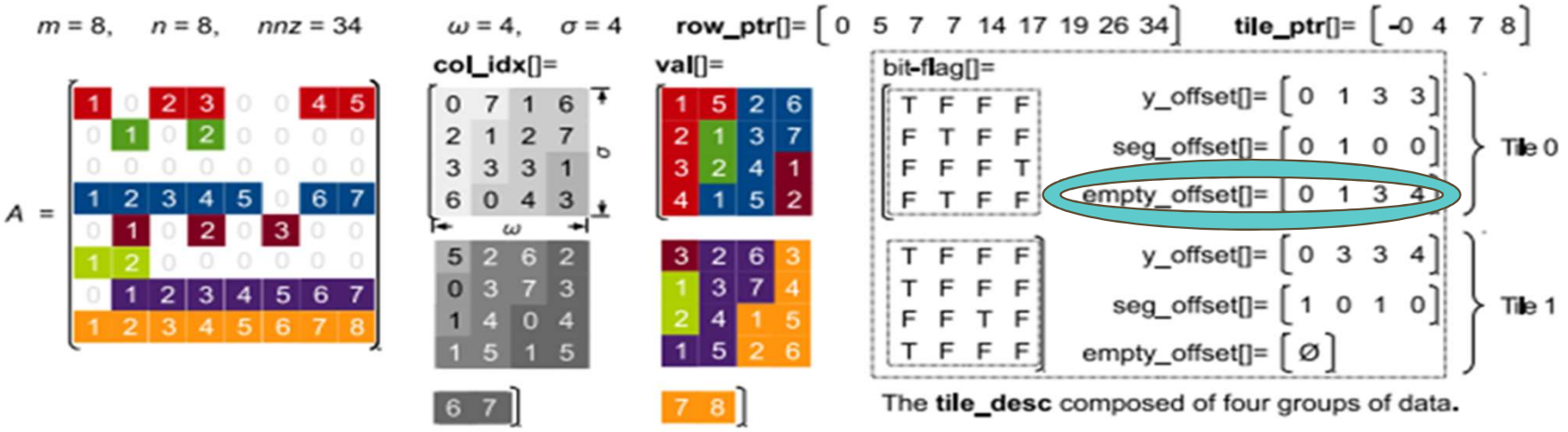


Figure 4: The CSR5 storage format of a sparse matrix A of size 8×8. The five groups of information include row_ptr, tile_ptr, col_idx, val and tile_desc.

CSR5格式及相关SpMV算法

大家都了解计算机存储是二进制数据，我们结合相关参数其生成的过程来探索相关参数信息空间大小，可以节省空间存储tile_desc数组。对于y_offset来说，其信息具有ωσ的上限。所以log2ωσ位对于y_offset中的每个条目都足够了。对于seg_offset信息量最多为ω，因此log2ω位就足够了。bit_flag为完整存储，存储区块的每列的σ个比特标志。因此，对于区块中的每一列，log2ωσ+log2ω+σ位二进制就足够存储。基于以上存储原理可以较好的实现压缩存储。

CSR5格式及相关SpMV算法

因为区块的信息（tile_ptr，tile_desc，col_idx和val）的所有计算彼此独立，所以它们可以同时执行。在GPU上，可为每个区块分配一堆线程（即nvidia GPU中的warp或AMD GPU中的wavefront，即GPU编程中分块block中很多thread）。在CPU和Xeon Phi上，我们使用OpenMP pragma来分配区块可用的x86核心。此外，区块内的列也是相互独立的，因此，可将GPU核心上的线程或x86核心上的SIMD通道分配给区块中的每个列。

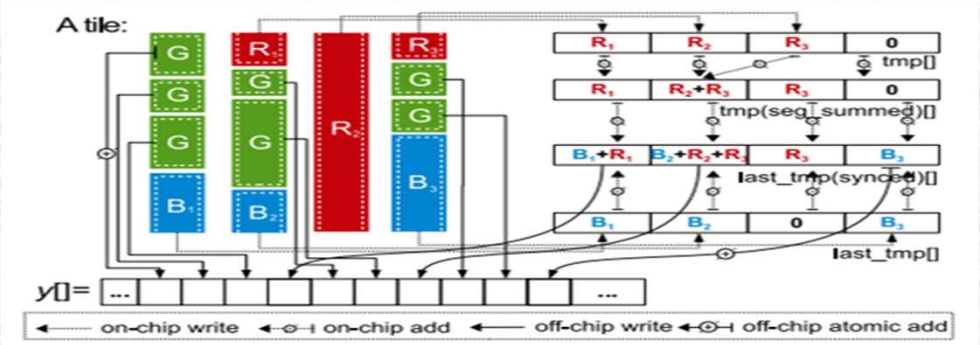


Figure 6: The CSR5-based SpMV in a tile. Partial sums of the green segments are directly stored to y. The red and the blue sub-segments require an extra segmented sum before issuing off-chip write.

Algorithm 8 The CSR5-based SpMV for the *tid*th tile.

```
1: MALLOC(*tmp, ω)
2: MEMSET(*tmp, 0)
3: MALLOC(*last_tmp, ω)
4: /*use empty_offset[y_offset[i]] instead of
   y_offset[i] for a tile with any empty rows*/
5: for i = 0 to ω − 1 in parallel do
6:     sum ← 0
7:     for j = 0 to σ − 1 do
8:         ptr ← tid × ω × σ + j × ω + i
9:         sum ← sum + val[ptr] × x[col_idx[ptr]]
10:        /*check bit_flag[i][j]*/
11:        if /*end of a red sub-segment*/ then
12:            tmp[i − 1] ← sum
13:            sum ← 0
14:        else if /*end of a green segment*/ then
15:            y[tile_ptr[tid] + y_offset[i]] ← sum
16:            y_offset[i] ← y_offset[i] +1
17:            sum ← 0
18:        end if
19:    end for
20:    last_tmp[i] ← sum //end of a blue sub-segment
21: end for
22: FAST_SEGMENTED_SUM(*tmp, *seg_offset)    ▷ Alg. 6
23: for i = 0 to ω − 1 in parallel do
24:    last_tmp[i] ← last_tmp[i] + tmp[i]
25:    y[tile_ptr[tid] + y_offset[i]] ← last_tmp[i]
26: end for
27: FREE(*tmp)
28: FREE(*last_tmp)
```

测试结果

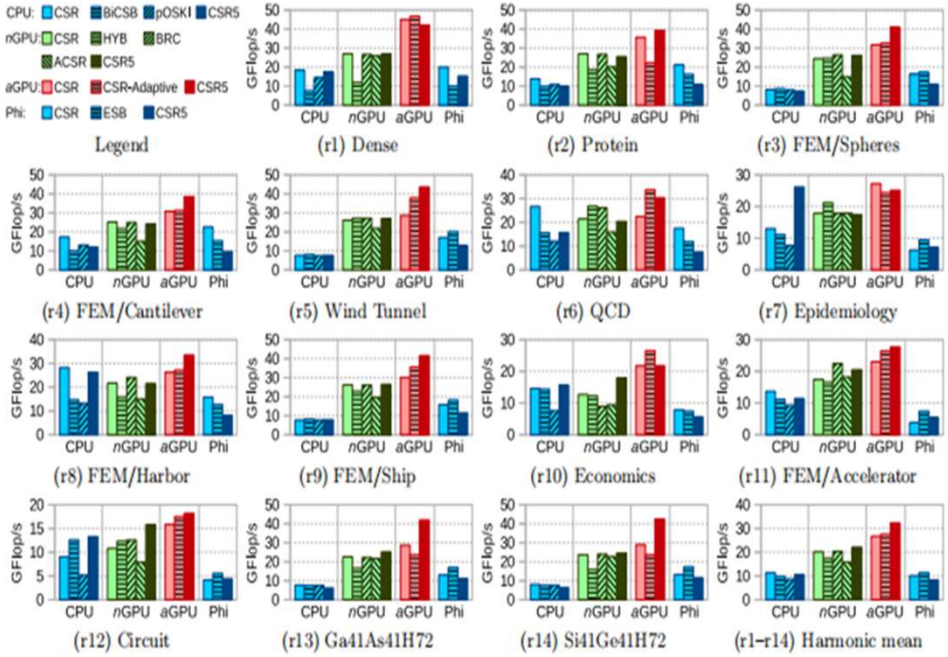


Figure 7: The SpMV performance of the 14 regular matrices. (*n*GPU=nVidia GPU, *a*GPU=AMD GPU)

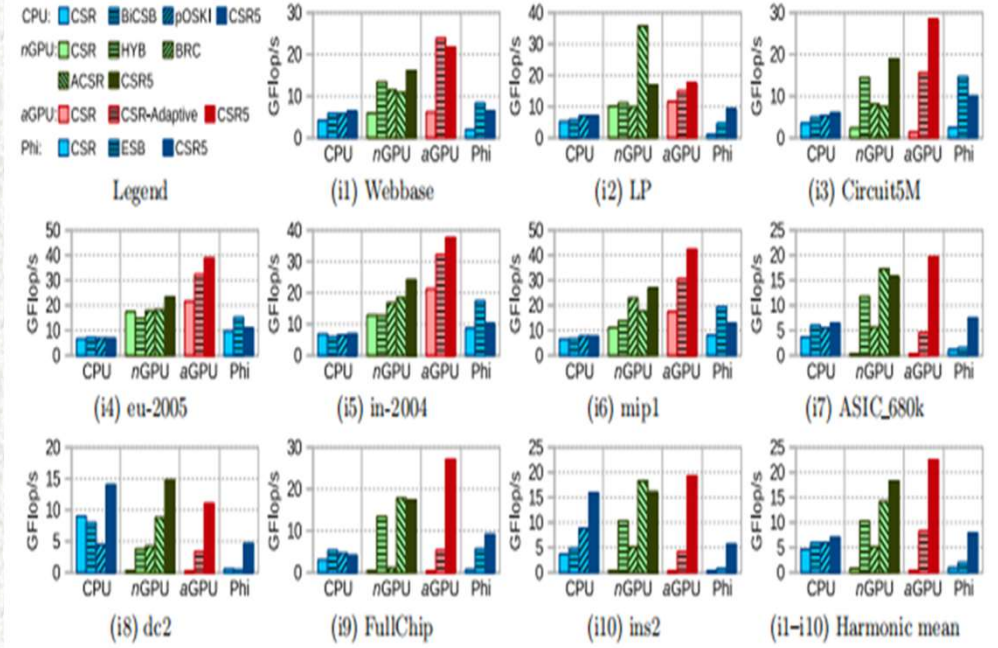


Figure 8: The SpMV performance of the 10 irregular matrices. (*n*GPU=nVidia GPU, *a*GPU=AMD GPU)

总结

1、从CSR到CSR5的格式转换的开销相当于几个SpMV操作开销，对于大规模稀疏矩阵计算来说格式转换开销可以忽略不计。

2、文章比较基于CSR5的SpMV算法四种主流的最先进的格式和算法处理器使用14个常规矩阵和10个不规则矩阵作为测试数据集。对于14个常规矩阵运算，CSR5格式可实现与之前工作相当或更好的性能。对于10个不规则矩阵，CSR5的平均性能提升为17.6%，28.5%，173.0%和293.3%（最高213.3%，153.6%，405.1%和943.3%）关于双插槽Intel CPU，nVidia GPU，AMD GPU和Intel Xeon Phi的最佳现有工作。在实际应用中，对于几十次迭代的求解器，CSR5格式更实用，因为它的格式转换开销很低。

谢谢您的聆听