

Weekly Research Progress Report

Student: 从 兴 Date: 2024/04/16 - 2023/05/01

List of accomplishments this week

- (1) 阅读论文《CSR5:An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication》
- (2) 阅读论文《IA-SpGEMM:An Input-aware Auto-tuning Framework for Parallel Sparse Matrix-Matrix Multiplication》
- (3) 完成初版专利申请书《基于分块矩阵结构的 GPU 自适应内存感知与优化技术》的编写
- (3) 推进 HYCOM 的工作
- (4) 改写 TileSpMV 的代码

Paper summary

Name: CSR5:An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication

Motivation: 目前，为了加速大规模计算，并行 SpMV 仍然需要用特定的数据存储格式和算法进行手工优化，这里面需要考虑在面向 SpMV 基本格式之间进行格式转换的成本。为了避免格式转换开销，现存的一些算法关注于通过行块方法或分段求和方法加速基于 CSR 的 SpMV。然而，行块方法会出现不可避免的负载不平衡，以至于在非规则矩阵上可能提供非常低的性能。分段求和的方法可以几乎避免负载不平衡，但是会因为使用了更多的全局同步和全局内存访问而导致较高的开销。因此，该论文提出了一种新的存储格式 CSR5，它可以在不需要格式转换的情况下，同时兼顾负载平衡和内存访问的优化。

Solution: 提出了 CSR5 存储格式，该格式实现了两个特点，一是该格式对稀疏结构不敏感，也就是说，该格式既适用于常规矩阵的高效 SpMV 算法也适用于不规则矩阵的高效 SpMV 算法。二是该格式转换的开销低，对于迭代次数较多的应用，该格式的转换开销可以忽略不计。

Related to us: 学习各个稀疏矩阵存储格式的优缺点，为之后论文的撰写提供思路。

Name: IA-SpGEMM: An Input-aware Auto-tuning Framework for Parallel Sparse Matrix-Matrix Multiplication

Motivation: 目前提出的 SpGEMM 算法几乎都局限于压缩的稀疏行格式 (CSR 格式), 并且没有很好的研究利用其他格式可能带来的性能增益, 而且, 能够为 SpGEMM 提供最佳性能的特定格式和算法也尚未确定。

Solution: 对特定格式的并行 SpGEMM 算法进行了研究, 提出了一种自适应的 SpGEMM 框架, 该框架可以根据输入矩阵的特性自动选择最佳的格式和算法。在实现自动选择的过程中, 设计了一个 MatNet 的神经网络, 该网络可以根据输入矩阵的特性预测最佳的格式和算法。

Related to us: 学习自动选择最佳格式和算法的方法, 为之后论文的撰写提供思路。

Work summary

1. CSR5 论文阅读

(1) 分段求和的方法

分段求和对基于 CSR 的 SpMV 可能更有吸引力, 因为它对 SIMD 比较友好, 对输入矩阵的稀疏结构不敏感, 从而克服了行块方法的缺点。分段求和对数组中每个分段中的条目执行求和运算。

一个段的第一个条目标记为 TRUE, 段内的其他条目标记为 FALSE。

Algorithm 2 Serial segmented sum operation.

```
1: function SEGMENTED_SUM(*in, *flag)
2:   length ← SIZEOF(*in)
3:   for i = 0 to length - 1 do
4:     if flag[i] = TRUE then
5:       j ← i + 1
6:       while flag[j] = FALSE && j < length do
7:         in[i] ← in[i] + in[j]
8:         j ← j + 1
9:       end while
10:    else
11:      in[i] ← 0
12:    end if
13:  end for
14: end function
```

图 1 分段求和的思想

通过分段求和, 可以使得每个段内的非零元素的个数都是基本相同的, 从而实现接近完美的负载均衡, 但是, 在 SpMV 计算中, 如果出现跨行的段或者每个行有多个段, 则需要用到同步机制, 这会造成一定的时间开销。

(2)CSR5 基本格式

为了实现具有任何稀疏结构的矩阵的近乎最佳负载平衡，首先将所有非零条目均匀分割到多个相同大小的二维 tile 中。因此，在执行并行 SpMV 操作时，计算核心可以消耗一个或多个二维 tile，并且核心的每个 SIMD 通道可以处理一个 tile 的一列。

然后 CSR5 格式的主要骨架只是一组二维 tile。CSR5 格式有两个调整参数： ω 和 σ ，其中 ω 是一个 tile 的宽度， σ 是 tile 的高度。将 tile 宽度 ω 设置为所使用处理器的 SIMD 执行单元大小，比如：设置 $\omega = 4$ 适用于具有 256 位 SIMD 单元的 CPU， $\omega = 32$ 适用于 nVidia GPU， $\omega = 64$ 适用于 AMD GPU，以及 $\omega = 8$ 对于具有 512 位 SIMD 单元的 Intel Xeon Phi。而 σ 的设置比较复杂，需要考虑的因素比较多，公式如下：

$$\sigma = \begin{cases} r & \text{if } \text{nnz}/\text{row} \leq r \\ \text{nnz}/\text{row} & \text{if } r < \text{nnz}/\text{row} \leq s \\ s & \text{if } s < \text{nnz}/\text{row} \leq t \\ u & \text{if } t < \text{nnz}/\text{row} \end{cases}$$

此外，还需要额外的信息来有效地计算 SpMV。对于每个 tile，引入一个 tile 指针 tile_ptr 和一个 tile 描述符 tile_desc。同时，将经典 CSR 格式的行指针 row_ptr、列索引 col_idx 和值 val 这三个数组直接集成。

唯一的区别是，每个完整块中的 col_idx 数据和 val 数据被就地调换（即，从行为主顺序到列为主顺序），以便从连续的 SIMD 通道进行合并内存访问。如果矩阵的最后一项没有填满一个完整的 2D Tile（即： $\text{nnz} \bmod (\omega\sigma) = 0$ ），它们就保持不变并丢弃它们的 tile_desc。

(3)tile_ptr 数组

```
Algorithm 4 Generating tile_ptr.  
1: for tid = 0 to p in parallel do  
2:   bnd ← tid × ω × σ  
3:   tile_ptr[tid] ← BINARY_SEARCH(*row_ptr, bnd) - 1  
4: end for  
5: for tid = 0 to p - 1 do  
6:   for rid = tile_ptr[tid] to tile_ptr[tid + 1] do  
7:     if row_ptr[rid] = row_ptr[rid + 1] then  
8:       tile_ptr[tid] ← NEGATIVE(tile_ptr[tid])  
9:     break  
10:    end if  
11:  end for  
12: end for
```

查找每个Tile中第一个非零元素所在的行位置。
如果某个Tile中包含空行，则该Tile的ptr设置为负数。

图 2 生成 tile_ptr 的算法

为了构建该数组，在 row_ptr 数组上二进制搜索每个 Tile 第一个非零条目的索引。如果相应 Tile 中包含任何空行，则将 tile_ptr 中的条目设置为其负值。因此，文章使用

无符号 32 位或 64 位整数作为 tile_ptr 数组的数据类型，有 1 位用于显式存储符号，31 或 63 位用于索引。

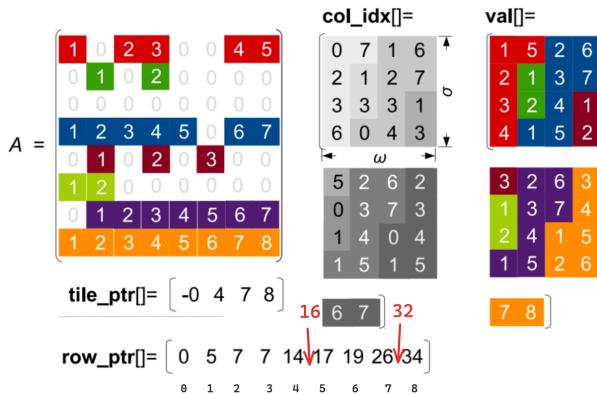


图 3 生成 tile_ptr 数组的示意图

(4) tile_desc 结构

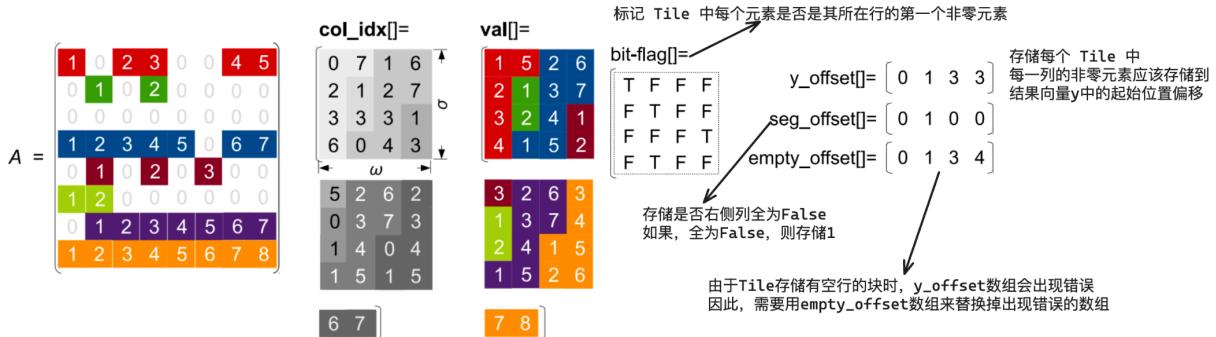


图 4 tile_desc 的示意图

- (1) 大小为 $\omega \times \sigma$ 的 bit_flag，用来标记 Tile 中每个元素是否是其所在行的第一个非零元素。
- (2) 大小为 ω 的 y_offset 用于存储每个 Tile 中每一列的非零元素应该存储到结果向量 y 中的起始位置偏移。
- (3) 大小为 ω 的 seg_offset ，它存储每列可以跳过的连续 FALSE 标记的数量，用于快速进行局部部分段求和计算。
- (4) 不固定大小的 $empty_offset$ ，当 Tile 包含一个或多个完全为空的行时，这些行的数据会被忽略，而 $empty_offset$ 提供了必要的索引信息，以确保这些空行的数据能正确地映射到结果向量 y 中的正确位置。

(5) 基于 CSR5 结构的 SpMV 计算

因为 2D Tile 的信息 (tile_ptr, tile_desc, col_idx 和 val) 的所有计算都彼此独立，它们可以并发执行。在 GPU 上，为每个 Tile 分配一组线程。在 CPU 和 Xeon Phi 上，使用

OpenMP pragma 将 Tile 分配给可用的 x86 核心。此外，在一个 Tile 内部，列也是相互独立的。因此，可以为每个列分配一个 GPU 核心上的线程或者 x86 核心上的 SIMD 通道。

Algorithm 8 The CSR5-based SpMV for the tid^{th} tile.

```

1: MALLOC(*tmp,  $\omega$ ) 第  $tid$  个 Tile 的计算
2: MEMSET(*tmp, 0)
3: MALLOC(*last_tmp,  $\omega$ ) 如果tile中有空行，就使用empty_offset代替掉offset
4: /*use empty_offset[y_offset[i]] instead of
y_offset[i] for a tile with any empty rows*/
5: for  $i = 0$  to  $\omega - 1$  in parallel do
6:   sum  $\leftarrow 0$  按列进行线程并行
7:   for  $j = 0$  to  $\sigma - 1$  do
8:     ptr  $\leftarrow tid \times \omega \times \sigma + j \times \omega + i$ 
9:     sum  $\leftarrow sum + val[ptr] \times x[col\_idx[ptr]]$ 
10:    /*check bit_flag[i][j]*/
11:    if /*end of a red sub-segment*/ then
12:      tmp[i-1]  $\leftarrow sum$  说明属于上一个部分的一部分
13:      sum  $\leftarrow 0$ 
14:    else if /*end of a green segment*/ then
15:      y[tile_ptr[tid] + y_offset[i]]  $\leftarrow sum$ 
16:      y_offset[i]  $\leftarrow y_offset[i] + 1$ 
17:      sum  $\leftarrow 0$  完整的一行，可以直接写入y值
18:    end if
19:  T... end for
20:  last_tmp[i]  $\leftarrow sum$  //end of a blue sub-segment
21: end for
22: FAST_SEGMENTED_SUM(*tmp, *seg_offset) > Alg. 6
23: for  $i = 0$  to  $\omega - 1$  in parallel do
24:   last_tmp[i]  $\leftarrow last\_tmp[i] + tmp[i]$ 
25:   y[tile_ptr[tid] + y_offset[i]]  $\leftarrow last\_tmp[i]$ 
26: end for
27: FREE(*tmp) 这里的y_offset[i]，如果第i列出现绿段，也就是有完整的行出现，y_offset[i]会自动加1
28: FREE(*last_tmp) 因此，最终会计算出正确的对应的y结果

```

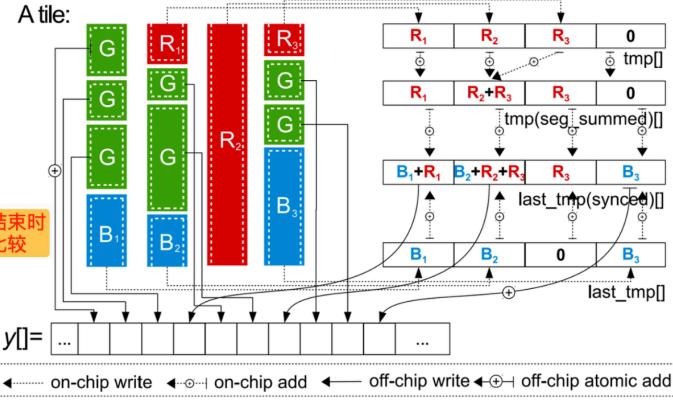


图 5 基于 CSR5 的 SpMV 算法

在运行基于 CSR5 的 SpMV 时，每个 Tile 中的列可以从 bit_flag 提取信息，并将其本地数据中段标记为三种颜色：

- (1) 红色表示从顶部未封闭子段
- (2) 绿色表示存在完全封闭段位于中间位置
- (3) 蓝色表示从底部未封闭子段。有一个例外情况是如果某一列既从顶部又从底部未封闭，则会被着以红色。

2. IA-SpGEMM 论文阅读

(1) 背景

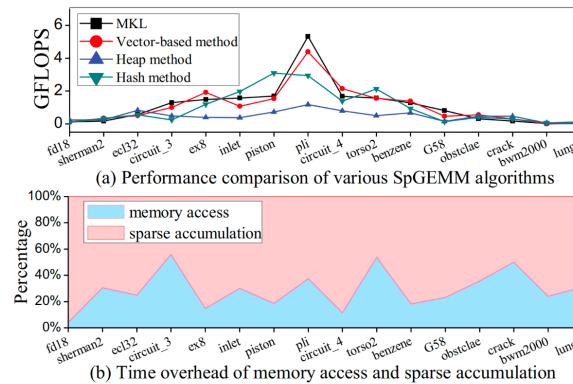


图 6 不同算法的性能比较以及开销

通过在 Intel CPU 上计算 $A * A^T$ 来比较各种算法的性能。很明显，不同的算法在不同的矩阵上提供它们的最佳性能，并且没有单一的算法在所有数据集的性能方面占主

导地位。同时，上图还展示了两个开销的来源：稀疏计算和内存访问。显然，内存访问占用了大量的执行时间。然而，该领域的研究很大程度上忽视了通过优化内存访问来提高性能的潜力，并且更倾向于继续为计算部分开发新的稀疏累积算法。在一定程度上，SpGEMM 类似于稀疏矩阵向量乘 (SpMV) 和稀疏三角解 (SpTRSV) 的不规则和间接存储器访问模式。SpMV 和 SpTRSV 的大部分研究都致力于通过挖掘经典存储格式来优化内存访问，并取得了有希望的结果。回到 SpGEMM，像 DIA、COO 和 ELL 这样的经典存储格式可以减少内存需求或加速向量架构上的内存访问，改变计算过程的顺序，甚至可以减少稀疏计算操作的数量。这项工作的另一个动机是探索几种经典存储格式对 SpGEMM 的影响。

(2) 三种 SpGEMM 算法

注：这里提到的三种算法，作者并没有在 GPU 中进行实现，仅仅实现在了 CPU 上。如下图所示。

注：在 TileGEMM 中，子块的存储结构只使用了 CSR 存储结构这一种。

	Method	Dominance		Percentage		Average Speedup	Speedup by "Ideal Tool"
		Best of all	Over BL.	Best of all	Over BL.		
Intel CPU	MKL (Baseline)	2874	-	35.07%	-	-	8.94x
	DIA method	491	1107	5.99%	13.51%	72.04x	
	COO method	283	506	3.45%	6.17%	7.63x	
	ELL method	1496	2879	18.26%	35.13%	9.92x	
	SPA vector-based	259	748	3.16%	9.13%	1.31x	
	Hash-based	2150	4307	26.24%	52.56%	6.37x	
	Heap-based	642	1951	7.83%	23.81%	6.21x	
AMD CPU	MKL (Baseline)	1708	-	20.96%	-	-	46.16x
	DIA method	745	1544	9.14%	18.94%	346.0x	
	COO method	342	586	4.20%	7.19%	8.20x	
	ELL method	1989	3044	24.40%	37.35%	32.96x	
	SPA vector-based	830	2529	10.18%	31.03%	1.58x	
	Hash-based	1757	2363	21.56%	28.99%	21.40x	
	Heap-based	779	1015	9.56%	12.45%	12.18x	
NVIDIA GPU	cuSPARSE(Baseline)	3827	-	51.07%	-	-	2.40x
	CUSP	208	769	2.78%	10.26%	6.27x	
	NSPARSE	3459	3525	46.16%	47.04%	3.71x	

没有设计针对 GPU 的算法啊！

图 7 为 CPU 开发的七个 SpGEMM 算法以及为 GPU 开发的三个 SpGEMM 算法

分别基于 DIA、COO 和 ELL 三种经典存储格式，作者提出了三种 SpGEMM 算法。这三种算法的主要思想是：首先，将输入矩阵转换为 DIA、COO 或 ELL 格式，然后使用矩阵乘法计算结果。如下图所示。

(3) IA-SpGEMM 框架

该文章为 SpGEMM 开发了一个输入感知的自动调优框架 (IA-SpGEMM)，以便在多个体系结构上选择最佳的格式和算法。

- 实线表示收集和培训阶段，虚线表示用户界面的执行流程。
- 收集阶段包括提取两种输入模式和所有算法的执行时间。

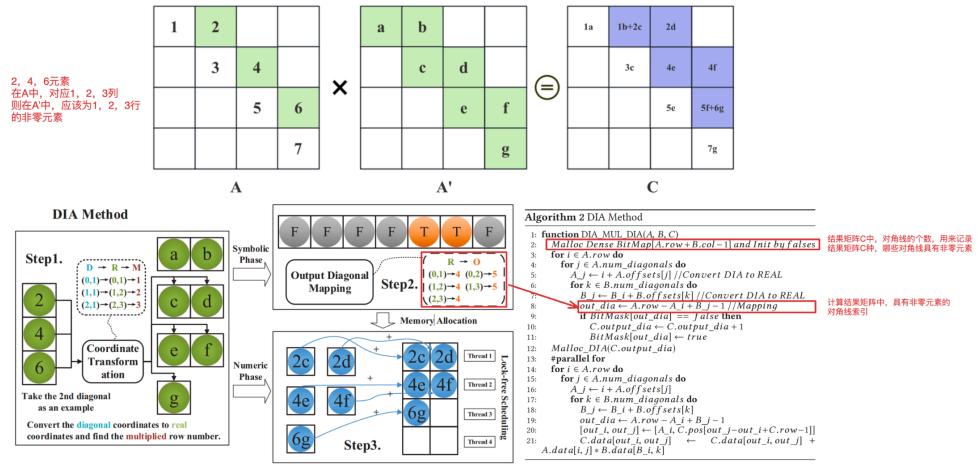


图 8 DIA 的 SpGEMM 算法

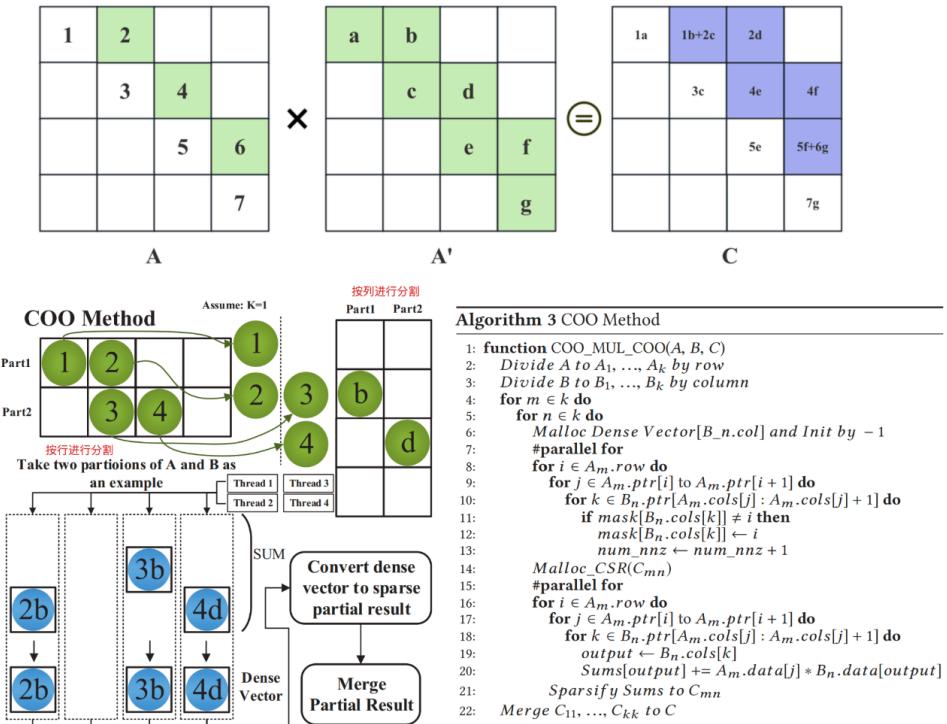


图 9 COO 的 SpGEMM 算法

- 训练阶段采用双向策略生成 MatNet 模型。

该框架考虑了 SpGEMM 内核中矩阵模式和机器配置对性能的影响，并通过数千个矩阵对进行评估。为了实现这一目标，需要一个学习模型，结合大量的矩阵模式，算法和机器配置，以找到最佳匹配解决方案。然而，对于一般算法来说，在大搜索空间中寻找最合适的解是一个挑战。因此，本文首先将自动校正问题转换为特征和图像分类问题，并选择一个优秀的卷积神经网络进行分类，以达到这个目的。

本文使用来自 SuitSparse Matrix Collection 的全部 2,726 个矩阵来构建 8000 多个矩阵乘法对，并提取 矩阵特征 和 密度表示 作为训练数据的输入。然后，收集 各种格式 和 算法的执行时间 作为训练数据的另一个输入。该方法结合矩阵特征和算法一起自动生成一个高精度的分类器。即，该方法，首先利用收集到的训练数据对神经网络 MatNet

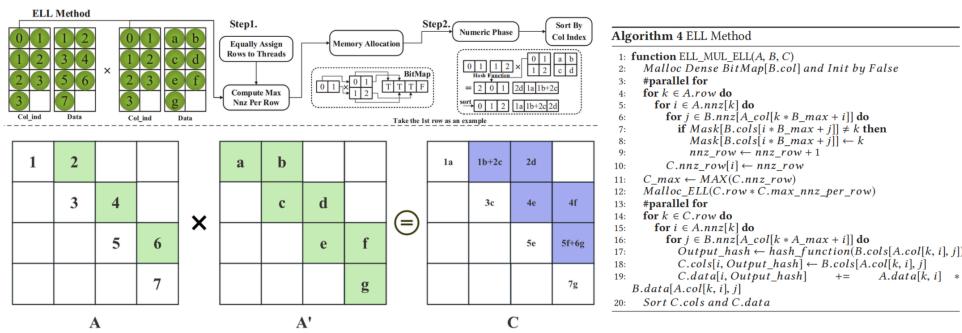


图 10 ELL 的 SpGEMM 算法

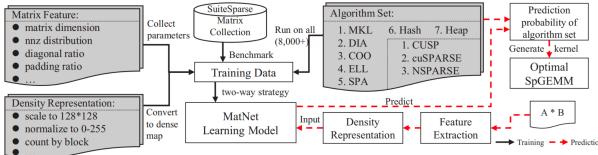


图 11 IA-SpGEMM 的框架图

进行训练，然后由预测部分指出每种算法生成最佳 SpGEMM 核的概率，最后选择出最佳的来执行。方便的是，IA-SpGEMM 提供了一个基于 CSR 格式的统一接口，使其具有可用性和可移植性。

(a) 矩阵特征

Table 3: Sparse features and description.

Feature	Description
row, col, nnz	the number of rows, columns and non-zero elements
nnz_ratio	the ratio of non-zero elements in CSR format
max, min, average	the maximum, minimum, and average numbers of non-zero elements
VAR	the variance of non-zero elements
dia_num	the number of diagonals in DIA format
dia_ratio	the number of diagonals divided by all diagonals
dia_pad, ell_pad	the ratio of padding data in DIA and ELL formats
CV	the coefficient of variation of non-zero elements

图 12 为训练从而提取的矩阵特征

前八个特性表示最常见的结构，包括行和列的数量，以及适合所有四种格式的非零元素。第 9 至第 11 个描述了 DIA 格式的对角线特征，包括对角线的数量和添加的零元素的填充率。第 12 个表示对齐内存访问的 ELL 格式的填充率。第 13 个特性代表了在中用于评估每行非零元素数量多样性。

(b) 密度表示

稀疏矩阵通常具有高稀疏性和不同大小，而卷积神经网络 (CNN) 通常需要固定大小的输入数据。

对于图像领域，一般的方法是缩小较大的像素或放大较小的像素来调整图像大小。该方法还可以用于将稀疏矩阵转换为小密度表示，以表示原始矩阵的粗粒度模式，并具有可接受的大小。

```

Algorithm 4 ELL Method
1: function ELL_MUL_ELL(A, B)
2:   Malloc Dense BitMap[B.col] and Init by False
3:   #parallel for
4:   for k ∈ A.col do
5:     for i ∈ A.nzl[k] do
6:       for j ∈ B.nzl[A.col[k]*B.max + i] do
7:         if Mask[B.col[i]*B.max + j] ≠ k then
8:           Mask[B.col[i]*B.max + j] ← k
9:           nzl_row ← nzl_row + 1
10:          C.nzl_row[i] ← nzl_row
11:          C.max ← MAX(C.nzl_row)
12:          Malloc_ELL(C.row * C.max_nnz_per_row)
13:          #parallel for
14:          for k ∈ C.row do
15:            for i ∈ A.nzl[k] do
16:              for j ∈ B.nzl[A.col[k]*A.max + i] do
17:                Output_hash ← hash_function(B.cols[A.col[k], i], j)
18:                C.cols[i, Output_hash] ← B.cols[A.col[k], i, j]
19:                C.data[i, Output_hash] += A.data[k, i] *
20:                  B.data[A.col[k], i, j]
Sort C.cols and C.data

```

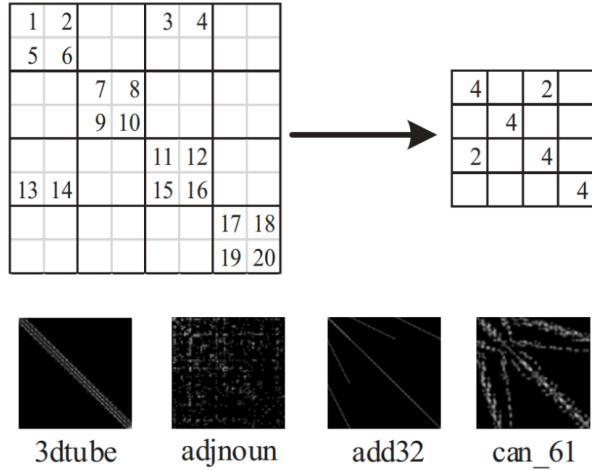


图 13 密度表示的转换过程

(c) MatNet 设计

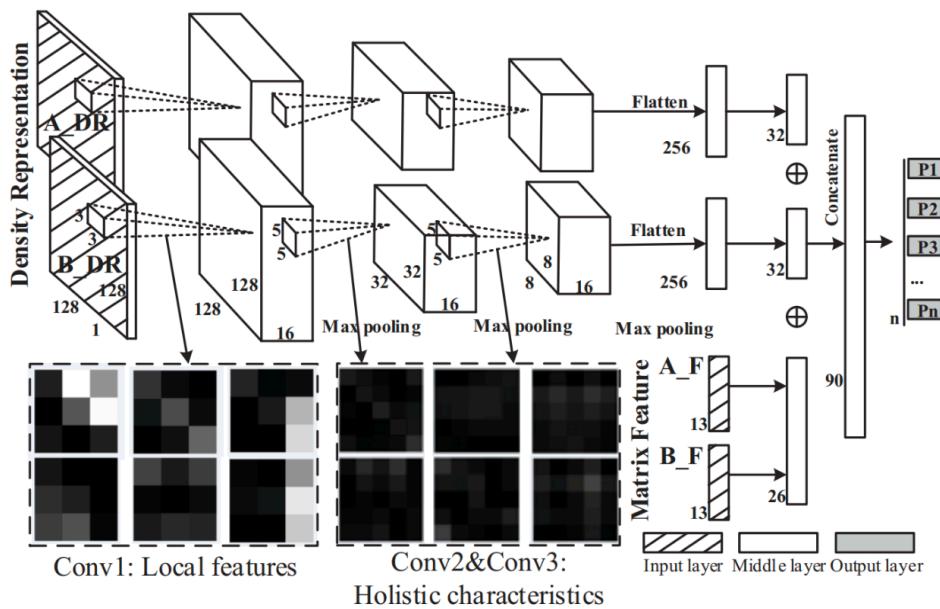


图 14 MatNet 模型

基于神经网络强大的分类能力，该文章设计了 MatNet 模型，将 CNN 和 FFNN 结合起来，提高了图像和特征同时分类的能力。如图 11 所示，这种结构由四个独立的输入组成，其中两个是矩阵 A 和 B 的密度表示，分别标记为 A_DR 和 B_DR，另一个是矩阵 A 和 B 的特征，分别标记为 A_F 和 B_F。这样，CNN 就可以通过使用 conv1 层和使用 conv2/conv3 层（一些核可视化）的整体特征来产生大量的过滤器来区分局部特征。

Next

- (1) 根据之前整理的前三年的论文，选取其中文章进行阅读。
- (2) 继续推进 HYCOM 的工作。
- (3) 继续推进 TileSpMV 的代码改写。