GPU高性能计算中SpMV性能优化学习报告

报告人：黄一智
指导老师：李仁发教授、刘彦老师
2017-04-28

论文一：Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors

- 来源：The International Conference for High Performance Computing, Networking, Storage and Analysis(SC’09 CCF 高性能计算A类会议)
- 引用量：620
- 作者：Nathan Bell and Michael Garland (NVIDIA Research)
- 论文创新点：根据不同存储格式的稀疏矩阵提出并设计相应的能很好适应面向吞吐量（throughput-oriented）的体系结构（不仅仅是GPU）的SpMV方法。
- SpMV形式：

$$y \leftarrow Ax+y$$

DIA(Diagonal)格式

- Data矩阵：矩阵的行表示稀疏矩阵的行，列表示稀疏矩阵中非全零对角线，非全零对角线中零元素可以用任意数值表示。
- Offset向量：表示稀疏矩阵A的每条对角线相对主对角线的偏移

$$A=\begin{pmatrix}1 & 7 & 0 & 0\\ 0 & 2 & 8 & 0\\ 5 & 0 & 3 & 9\\ 0 & 6 & 0 & 4\end{pmatrix} \qquad data=\begin{pmatrix}* & 1 & 7\\ * & 2 & 8\\ 5 & 3 & 9\\ 6 & 4 & *\end{pmatrix} \qquad offset=\begin{bmatrix}-2 & 0 & 1\end{bmatrix}$$

DIA(Diagonal)格式CUDA优化方法：

- data数据按列顺序存储
- 每一个row分配给一个线程
- 还原矩阵A的row和col：

$$row = threadid$$

$$col = row + offset[i](i \in [0, dia\_num) \&\& col \geq 0 \&\& col \leq c$$

- 每一个线程进行对角线总数

次乘加运

$$data = [* \quad * \quad 5 \quad 6$$

	线程0	线程1	线程2	线程3
迭代0	*	*	5	6
迭代1	1	2	3	4
迭代2	7	8	9	*

```
__global__ void
spmv_dia_kernel(const int num_rows,
                const int num_cols,
                const int num_diags,
                const int   * offsets,
                const float * data,
                const float * x,
                      float * y)
{
    int row = blockDim.x * blockIdx.x + threadIdx.x;

    if(row < num_rows){
        float dot = 0;

        for(int n = 0; n < num_diags; n++){
            int   col = row + offsets[n];
            float val = data[num_rows * n + row];

            if(col >= 0 && col < num_cols)
                dot += val * x[col];
        }

        y[row] += dot;
    }
}
```

针对CUD

因此warp内

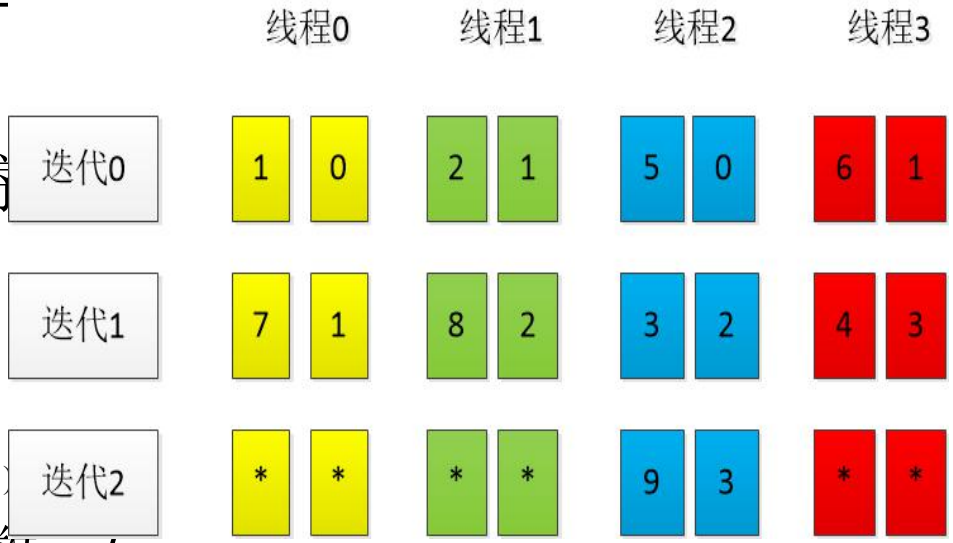
的线程可

ELL(ELLPACK)格式

- Data矩阵：按顺序保存稀疏矩阵每一行中的非零元素到矩阵中，非零元素最多的非零行的列数决定data矩阵的列，data矩阵中其余行未填满数据可以用特定数据填满。
- Indices矩阵：存放data矩阵中元素在稀疏矩阵中的对应列号

$$A=\begin{pmatrix}1 & 7 & 0 & 0\\ 0 & 2 & 8 & 0\\ 5 & 0 & 3 & 9\\ 0 & 6 & 0 & 4\end{pmatrix} \qquad data=\begin{pmatrix}1 & 7 & *\\ 2 & 8 & *\\ 5 & 3 & 9\\ 6 & 4 & *\end{pmatrix} \qquad indices=\begin{pmatrix}0 & 1 & *\\ 1 & 2 & *\\ 0 & 2 & 3\\ 1 & 3 & *\end{pmatrix}$$

ELL(ELLPACK)格式CUDA优化方法



- Data和indices数据均按列顺序
- 按行分配线程
- 还原矩阵A的row和col：

$$row = threadid$$

$$col = indices[i * row\_num + row](i \in [0, col\_num))$$

- 每一个线程执行data矩阵列数col_num次乘加操作

```
__global__ void
spmv_ell_kernel(const int num_rows,
                const int num_cols,
                const int num_cols_per_row,
                const int   * indices,
                const float * data,
                const float * x,
                      float * y)
{
    int row = blockDim.x * blockIdx.x + threadIdx.x;

    if(row < num_rows){
        float dot = 0;

        for(int n = 0; n < num_cols_per_row; n++){
            int   col = indices[num_rows * n + row];
            float val = data[num_rows * n + row];

            if(val != 0)
                dot += val * x[col];
        }
```

CSR(Compressed Sparse Row)格式

- Data数组：保存非零元素数值
- Indices数组：保存data数组中对应数据在稀疏矩阵中的列号
- Ptr数组：保存稀疏矩阵每行第一个非零元素在data数组中的索引，最后一项是非零元素总数(因此有row+1个元素)

$$A=\begin{pmatrix}1 & 7 & 0 & 0\\ 0 & 2 & 8 & 0\\ 5 & 0 & 3 & 9\\ 0 & 6 & 0 & 4\end{pmatrix}$$

$$data=\begin{bmatrix}1 & 7 & 2 & 8 & 5 & 3 & 9 & 6 & 4\end{bmatrix}$$

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

$$ptr=\begin{bmatrix}0 & 2 & 4 & 7 & 9\end{bmatrix}$$

CSR格式CUDA优化方法一：标量(Scalar)内核

- 按矩阵
- 问题：

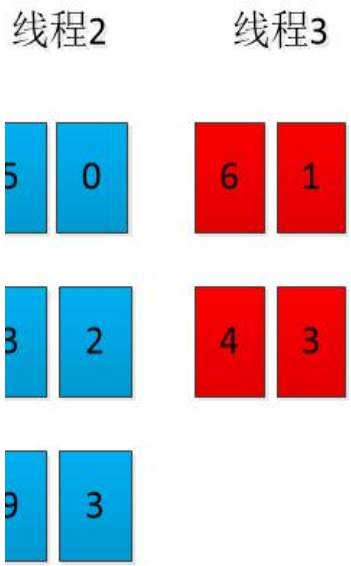
1、虽然
ELL类似，
格式却不
连续访问

data =

indices =

2、由于
却导致某

```
__global__ void
spmv_csr_scalar_kernel(const int num_rows,
                       const int   * ptr,
                       const int   * indices,
                       const float * data,
                       const float * x,
                             float * y)
{
    int row = blockDim.x * blockIdx.x + threadIdx.x;

    if(row < num_rows){
        float dot = 0;

        int row_start = ptr[row];
        int row_end   = ptr[row+1];

        for (int jj = row_start; jj < row_end; jj++)
            dot += data[jj] * x[indices[jj]];

        y[row] += dot;
    }
}
```



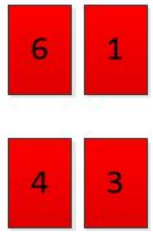
节约，
程长短不一，最终导致一个warp中可能多个线程出现空等。

CSR格式CUDA优化方法二：矢量(Vector)内核

- 按行
- 优点
非连续
- 缺点
操作非
空等的
分配给
数据块
个开销

```
__global__ void
spmv_csr_vector_kernel(const int num_rows,
                       const int   * ptr,
                       const int   * indices,
                       const float * data,
                       const float * x,
                             float * y)
{
    __shared__ float vals[];

    int thread_id = blockDim.x * blockIdx.x + threadIdx.x;  // global thread index
    int warp_id   = thread_id / 32;                         // global warp index
    int lane      = thread_id & (32 - 1);                   // thread index within the warp

    // one warp per row
    int row  = warp_id;

    if (row < num_rows){
        int row_start = ptr[row];
        int row_end   = ptr[row+1];

        // compute running sum per thread
        vals[threadIdx.x] = 0;
        for(int jj = row_start + lane; jj < row_end; jj += 32)
            vals[threadIdx.x] += data[jj] * x[indices[jj]];

        // parallel reduction in shared memory
        if (lane < 16) vals[threadIdx.x] += vals[threadIdx.x + 16];
        if (lane <  8) vals[threadIdx.x] += vals[threadIdx.x +  8];
        if (lane <  4) vals[threadIdx.x] += vals[threadIdx.x +  4];
        if (lane <  2) vals[threadIdx.x] += vals[threadIdx.x +  2];
        if (lane <  1) vals[threadIdx.x] += vals[threadIdx.x +  1];

        // first thread writes the result
        if (lane == 0)
            y[row] += vals[threadIdx.x];
    }
}
```

warp3



COO(Coordinate)格式

- Data数组：按行列顺序保存稀疏矩阵中非零元素值
- Row数组：保存data数组中元素在稀疏矩阵中的行坐标
- Col数组：保存data数组中元素在稀疏矩阵中的列坐标

$$A=\begin{pmatrix}1 & 7 & 0 & 0\\ 0 & 2 & 8 & 0\\ 5 & 0 & 3 & 9\\ 0 & 6 & 0 & 4\end{pmatrix}$$

$$data=\begin{bmatrix}1 & 7 & 2 & 8 & 5 & 3 & 9 & 6 & 4\end{bmatrix}$$

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

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

总结

- 本文主要讨论的是如何具体在cuda上优化SpMV的方法，也讨论相关方法和不同存储格式的SpMV的优劣。

- 存储结构上的优缺点带来了适应领域的不同，如：DIA适用全零对角线多的情况，ELL适用于最大非零行和平均非零行相差不大的矩阵等。针对稀疏矩阵的不同，采用不同结构及优化方法会得出一个求最优解的问题。

论文二：Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling

- 来源：IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS(TPDS),2014
- 作者：Kenli Li & Wangdong Yang & Keqin Li
- 论文创新点：提出了一种概率建模方法，建立不同格式SpMV的时间开销模型。通过这些开销模型可以计算出输入为稀疏矩阵A、向量x和向量y的情况下的各格式时间开销用以分析并获得最佳存储格式。

一、建立稀疏矩阵的PMF(Probability Mass Function)

- 对于稀疏矩阵$A_{N\times M}$，有随机变量$X \in \{0,1,\dots,M\}$表示每行的非零元素个数。定义集合$B = \{b_0, b_1, b_2 \dots, b_M\}$，其中$b_i$表示包含$i$个非零元素的行数：

$$\sum_{i=0}^{M} b_i = N$$

$\{X = i\}$的概率为：

$$P(X = i) = p_i = b_i/N$$

$$p_i \geq 0$$

并得出一些相应概率公式：

$$\sum_{i=0}^{M} p_i = \sum_{i=0}^{M} b_i/N = 1$$

$$P(X \leq K) = \sum_{i=0}^{K} p_i \quad P(X \geq K) = \sum_{i=K+1}^{M} p_i$$

期望公式：

$$E(X) = \sum_{i}^{M} i \times p_i$$

$$E(X = k|X \leq K) = \begin{cases} p_k / \sum_{i=0}^{K} p_i, (k \leq K) \\ 0, (k > K) \end{cases}$$

$$E(X = k|X > K) = \begin{cases} p_k / \sum_{i=K+1}^{M} p_i, (k > K) \\ 0, (k \leq K) \end{cases}$$

稀疏矩阵非零元素数目NNZ相关：

$$NNZ = E(X) \times N$$

$$NNZ(X \leq K) = E(X|X \leq K) \times N \times P(X \leq K)$$

$$NNZ(X > K) = E(X|X > K) \times N \times P(X > K)$$

二、对不同格式的存储空间分析

$$S_{COO} = S_s \times NNZ + 2 \times S_i \times NNZ$$
$$S_{CSR} = S_s \times NNZ + S_i \times NNZ + S_i \times (N + 1)$$
$$S_{ell} = S_s \times N \times K + S_i \times N \times K$$

三、时间开销建模

总时间：

$$T = DDT + CT$$

数据传输时间：

$$DDT = \frac{data\ size}{Bandwith}$$

计算时间：

$$CT = AM + CTC_m + CTC_a$$

仿存时间(DS是存在显存上的数据，RW是连续数据的长度，CR是显存的时钟频率)：

$$AM = \left\lceil \frac{DS}{RW} \right\rceil \times \frac{1}{CR}$$

核上乘法操作时间(C为流处理器个数，FS为单精度乘法或加法操作的速度)：

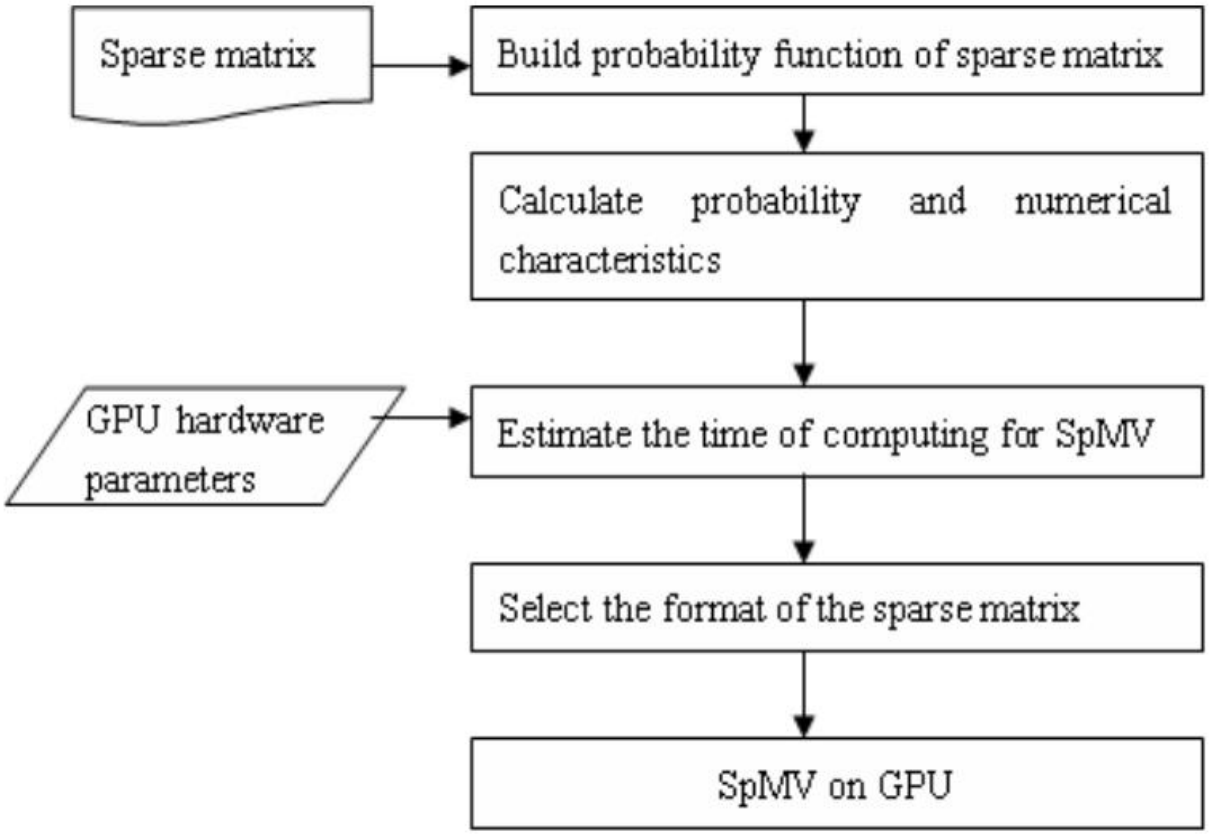
$$CTC_m = \frac{NNZ}{C \times F_s}$$

核上加法操作时间(w是warp中的线程数)：

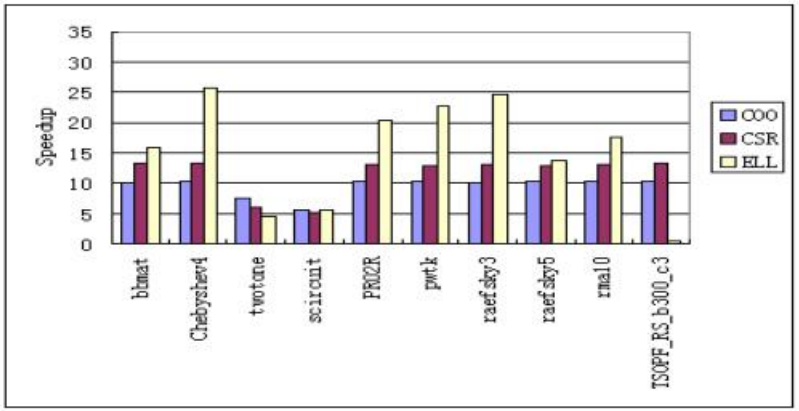
$$CTC_a = \frac{N \times W}{C} \times \left\lceil \frac{E(X)}{W} \right\rceil \times \frac{1}{F_S}$$

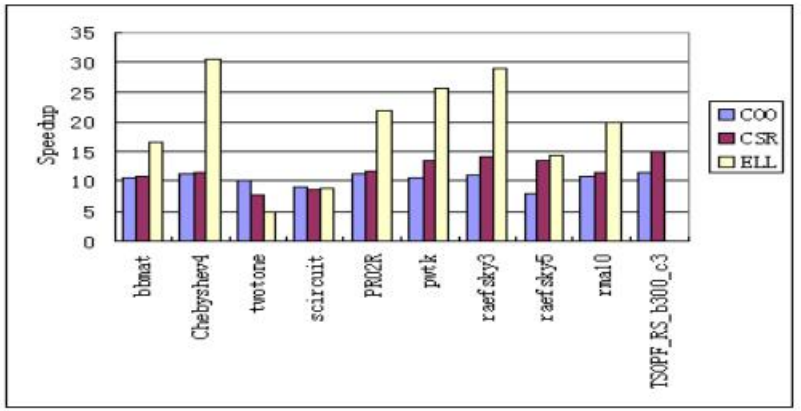
优化流程

- 建立PMF
- 计算概率和相关参
- 计算各个格式的时
- 选取最佳格式
- SpMV的CUDA计算

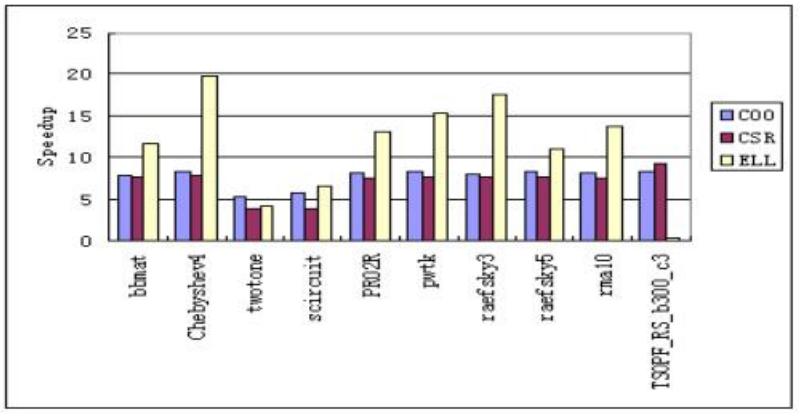


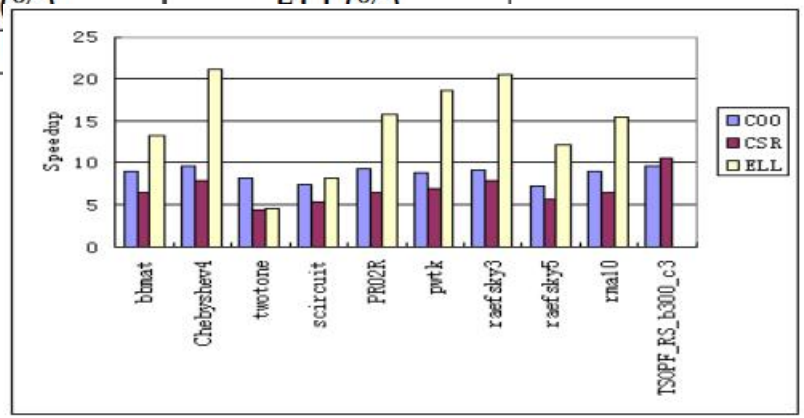
评价方式





的平均绝对值差不超过20%）





raefsky5	28.46	15.88	-3.40	34.29	-5.22	-8.40
rma10	-6.52	-10.10	13.78	14.33	-11.99	-10.46
TSOPF_RS_b300_c3	-10.69	-13.81	-11.44	-11.34	NA	NA

总结

- GPU对于这篇文章仅仅起到的是一个背景作用，更重要的是教给我们一种解决问题的方法，用概率模型去估计和分析系统性能，这个系统