SpMV并行优化学习报告

报告人：黄一智
指导老师：李仁发教授、刘彦老师
2017-05-19

Merge-based Parallel Sparse Matrix-Vector Multiplication

- 论文来源：The International Conference for High Performance Computing, Networking, Storage and Analysis(SC'16 CCF A类会议)
- 论文作者：Duane Merrill and Michael Garland (NVIDIA Corporation Santa Clara, CA，其中Michael Garland: lead the Programming Systems and Applications Research Group)
- 论文创新点：利用Merge-Path算法并行化CSR编码格式的SpMV，保证了线程任务负载均衡的同时，依然能有效并行。
- 论文贡献：提出算法、充分实验和工程级代码

论文背景

- 大量针对体系结构的自定矩阵结构与特定算法要求预处理，在应用场景下这些预处理带来了一些实际成本。

- 实际应用中，很少维护自定义编码的稀疏矩阵，基本都采用通用编码格式，如：CSR(Intel MKL and NVIDIA cuSPARSE)

问题提出

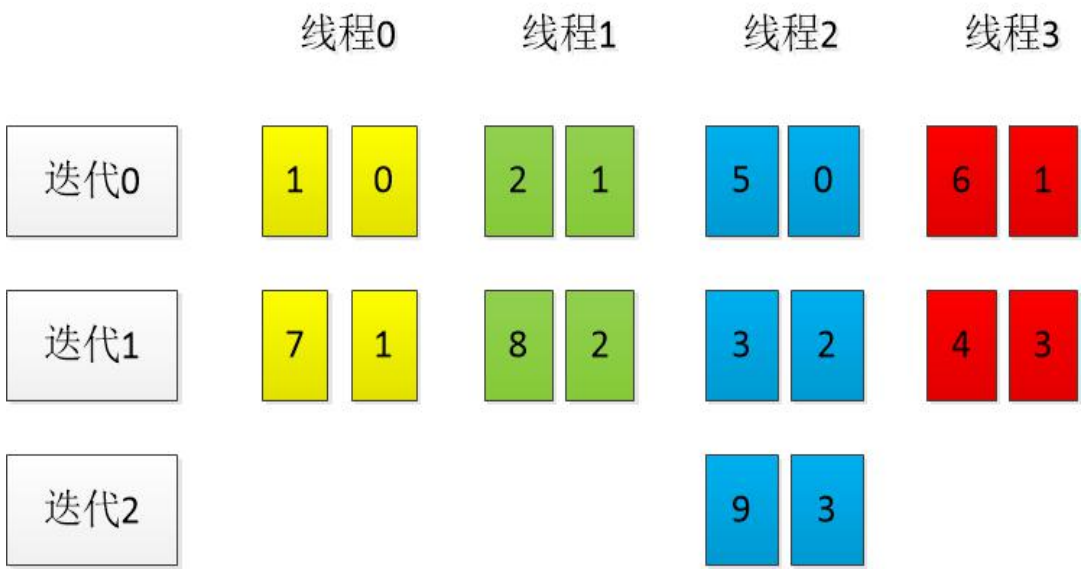
- CSR编码格式，在直接并行时会因为非零行的行宽比率变化而产生性能变化，极端行宽会导致性能急剧退化（负载不均衡所致）。以按矩阵行分配线程的标量内核法为例：

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

$$data=[1\quad 7\quad 2\quad 8\quad 5\quad 3\quad 9\quad 6\quad 4]$$

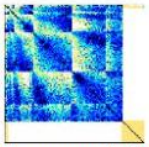
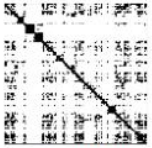
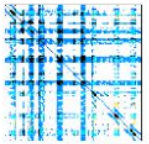
$$indices=[0\quad 1\quad 1\quad 2\quad 0\quad 2\quad 3\quad 1\quad 3]$$

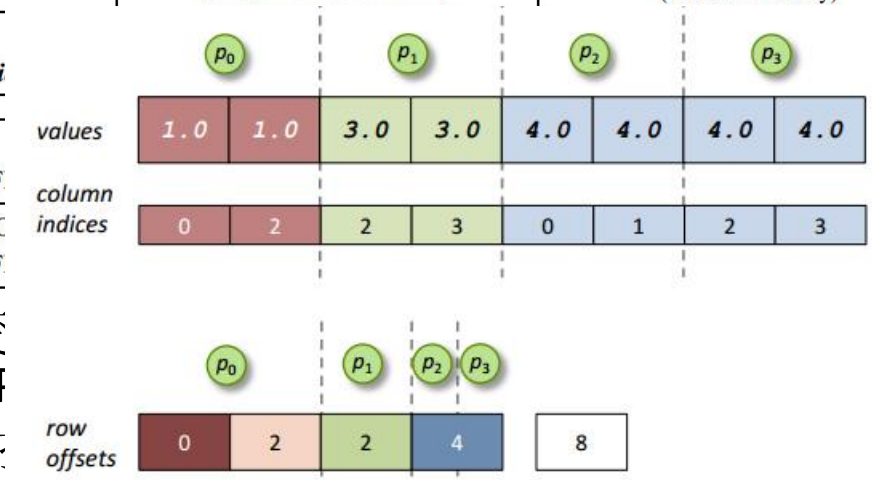
$$row\_offset=[0\quad 2\quad 4\quad 7\quad 9]$$



问题提出：作者的实验论证

TABLE 1: Relative consistency of double-precision CsrMV performance among similarly-sized matrices from the University of Florida Sparse Matrix Collection [7], evaluated using two Intel Xeon E5-2690 CPU processors (24-core each) and one NVIDIA Tesla K40 GPU

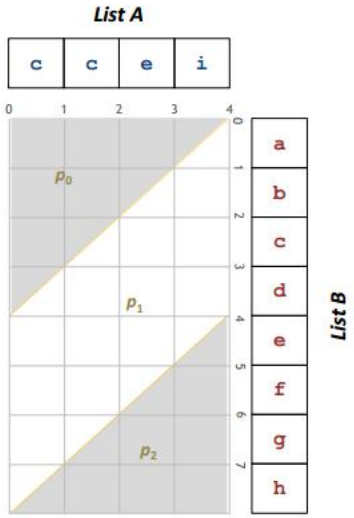
		 *thermomech_dK* (temperature deformation)	 *cnr-2000* (Web connectivity)	 *ASIC_320k* (circuit simulation)
Number of nonzeros (nnz)				2,635,364
Row-length coefficient of vari				61.4
CPU x2	*MKL* (GFLOPs)			11.8
	Merge-based (GF			**23.2**
GPU x1	*cuSPARSE* (GFLC			0.12
	Merge-based (GF			**14.1**

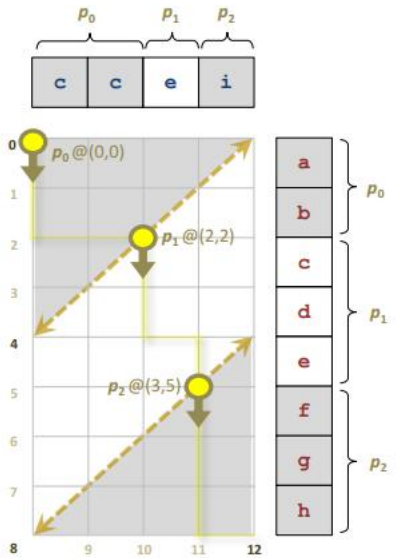


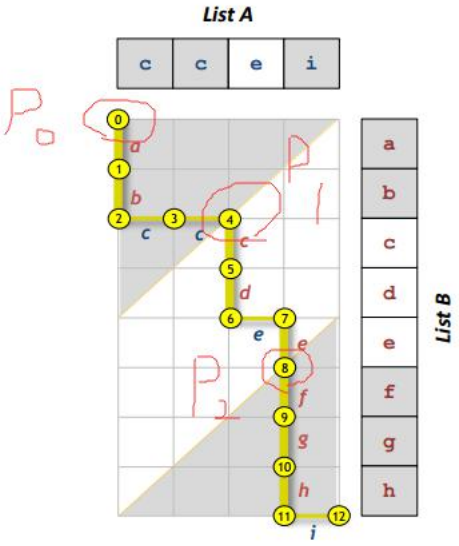
- 随着行长变 行SpMV性能都有所下降，GF
- 原因：CPU 任务少的线程也会在完成之后调度去进行下一个任务。GPU按warp调度(当前是32线程)，极端情况下如果一个warp中某个线程负载特别重，其余31个线程都要等待这一个线程完成。
- 问题：为何不均衡分配数据给线程？
- 难度：如何有效解决分配数据不足一行或者超过一行？(如 Nonzero-spliting)

并行Merge-path算法

- 两个已经排好序的向量A B，根据对角线k进行合并，合并时可以找到节点$(i,j), i+j=k$使得A_i大于B_j之前所有的元素。
- 这个节点(i,j)就是调度节点。







CSR对于Merge-path

- CSR的row_offset向量表示稀疏矩阵的每行的第一个非零元素在value数组中的偏移位置，因此row_offset一定是从小到大有序的。可以作为Merge-path中的A向量。
- CSR的value和column_indices数组的索引也是有序的，而且和row_offset值有关，因此可以作为Merge-path中的B向量。
- k为分配给每个线程的数据量

$$\begin{pmatrix} 1.0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 3.0 & 3.0 \\ 4.0 & 4.0 & 4.0 & 4.0 \end{pmatrix}$$

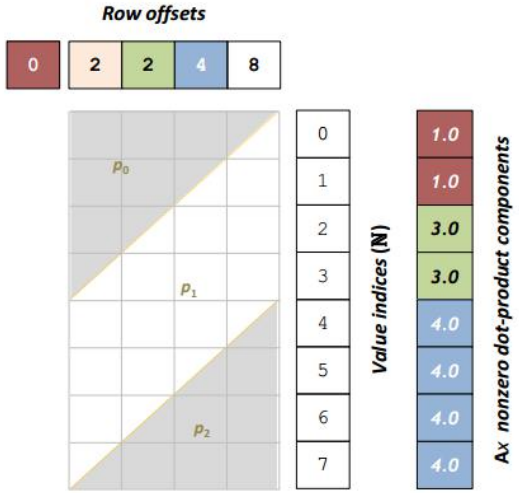
$$value = [1.0 \quad 1.0 \quad 3.0 \quad 3.0 \quad 4.0 \quad 4.0 \quad 4.0 \quad 4.0]$$

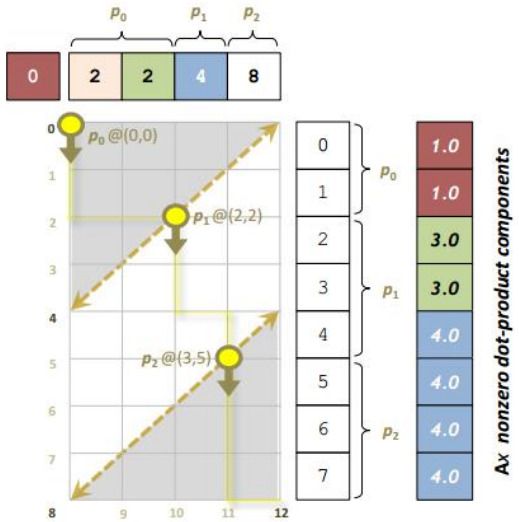
$$column\_indices = [0 \quad 2 \quad 2 \quad 3 \quad 0 \quad 1 \quad 2 \quad 3]$$

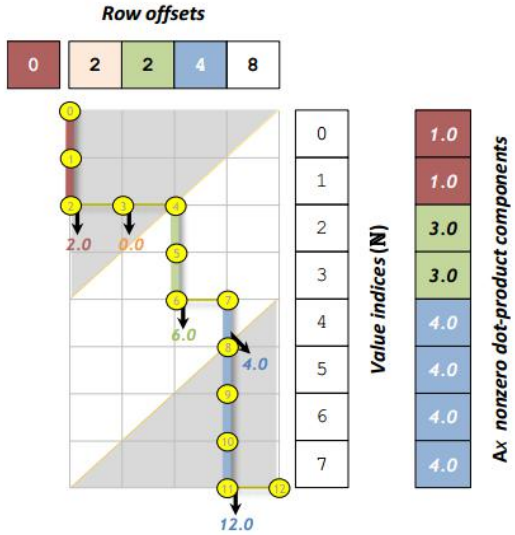
$$row\_offset = [0 \quad 2 \quad 2 \quad 4 \quad 8]$$

A 2 2 4 8

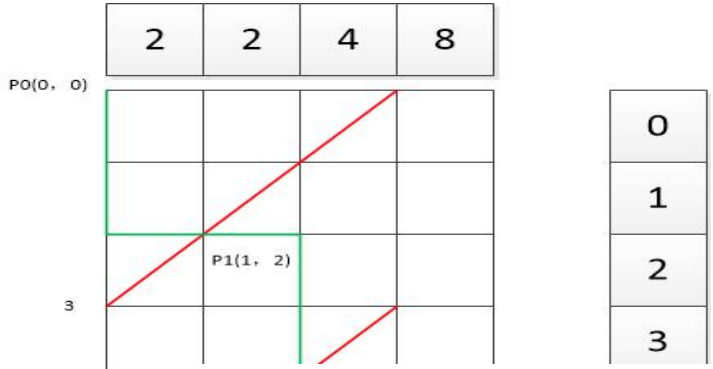
B
0
1
2
3
4
5
6
7


Row offsets
0
2
2
4
8
p_0
p_1
p_2
0
1
2
3
4
5
6
7
Value indices (N)
1.0
1.0
3.0
3.0
4.0
4.0
4.0
Ax nonzero dot-product components

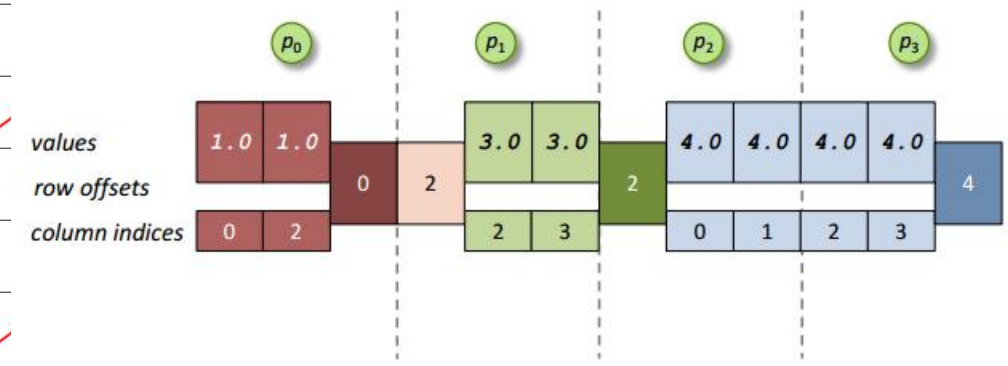

p_0
p_1
p_2
0
2
2
4
8
p_0 @(0,0)
p_1 @(2,2)
p_2 @(3,5)
0
1
2
3
4
5
6
7
8
9
10
11
12
p_0
p_1
p_2
1.0
1.0
3.0
3.0
4.0
4.0
4.0
Ax nonzero dot-product components


Row offsets
0
2
2
4
8
2.0
0.0
6.0
4.0
12.0
0
1
2
3
4
5
6
7
Value indices (N)
1.0
1.0
3.0
3.0
4.0
4.0
4.0
Ax nonzero dot-product components

3线程 $k = 4$


2
2
4
8
PO(O, O)
P1(1, 2)
3
6
9
0
1
2
3

4线程 $k = 3$


p_0
p_1
p_2
p_3
values
row offsets
column indices
1.0
1.0
3.0
3.0
4.0
4.0
4.0
4.0
0
2
2
4
0
2
2
3
0
1
2
3

ALGORITHM 2. The parallel merge-based CsrMV algorithm (C++ OpenMP)

Input: Number of parallel threads, CSR matrix **A**, dense vector x
Output: Dense vector y such that $y \leftarrow \mathbf{A}x$

```
1   void OmpMergeCsrmv(int num_threads, const CsrMatrix& A, double* x, double* y)
2   {
3       int* row_end_offsets = A.row_offsets + 1;                                  // Merge list A: row end-offsets
4       CountingInputIterator<int> nz_indices(0);                                  // Merge list B: Natural numbers(NZ indices)
5       int num_merge_items = A.num_rows + A.num_nonzeros;                         // Merge path total length
6       int items_per_thread = (num_merge_items + num_threads - 1) / num_threads;  // Merge items per thread
7
8       int row_carry_out[num_threads];
9       double value_carry_out[num_threads];
10
11      // Spawn parallel threads
12      #pragma omp parallel for schedule(static) num_threads(num_threads)
13      for (int tid = 0; tid < num_threads; tid++)
14      {
15          // Find starting and ending MergePath coordinates (row-idx, nonzero-idx) for each thread
16          int diagonal                = min(items_per_thread * tid, num_merge_items);
17          int diagonal_end            = min(diagonal + items_per_thread, num_merge_items);
18          CoordinateT thread_coord    = MergePathSearch(diagonal, row_end_offsets, nz_indices,
19                                            A.num_rows, A.num_nonzeros);
20          CoordinateT thread_coord_end = MergePathSearch(diagonal_end, row_end_offsets, nz_indices,
21                                            A.num_rows, A.num_nonzeros);
22
23          // Consume merge items, whole rows first
24          double running_total = 0.0;
25          for (; thread_coord.x < thread_coord_end.x; ++thread_coord.x)
26          {
27              for (; thread_coord.y < row_end_offsets[thread_coord.x]; ++thread_coord.y)
28                  running_total += A.values[thread_coord.y] * x[A.column_indices[thread_coord.y]];
29
30              y[thread_coord.x] = running_total;
31              running_total = 0.0;
32          }
33
34          // Consume partial portion of thread's last row
35          for (; thread_coord.y < thread_coord_end.y; ++thread_coord.y)
36              running_total += A.values[thread_coord.y] * x[A.column_indices[thread_coord.y]];
37
38          // Save carry-outs
39          row_carry_out[tid] = thread_coord_end.x;
40          value_carry_out[tid] = running_total;
41      }
42
43      // Carry-out fix-up (rows spanning multiple threads)
44      for (int tid = 0; tid < num_threads - 1; ++tid)
45          if (row_carry_out[tid] < A.num_rows)
46              y[row_carry_out[tid]] += value_carry_out[tid];
47  }
```

- 16、17行根据每个线程分配的数据数目求出k值
- 18、19行根据merge算法求出每个线程的(i, j)
- 求出区域内SpMv

实验：

- 数据集：University of Florida Sparse Matrix Collection(https://www.cise.ufl.edu/research/sparse/matrices, 8496个实际的稀疏矩阵，共254GB)
- 实验环境：多核CPU、GPU、NUMA(非统一内存访问架构)
- 比较对象：Intel MKL、NVIDIA cuSPARSE csrMV、CSB、HYB、pOSKI、yaSpMV
- 实验项目：峰值实验、预处理开销、内核性能一致性、相对加速比比较、矩阵数据吞吐量

总结：

- 以应用为导向的研究：以应用为导向的算法、以应用为导向的编码(https://github.com/dumerrill/merge-spmv.git)。
- 合理的将自身问题贴合到已有算法。
- 充分的实验论证。