Report: Improving Parallel Sparse Matrix-vector Multiplication

Kaisar Barlybay, Olzhas Sanatbek

December 20, 2022

1 Introduction & Summary

Sparse Matrix-vector Multiplication (SMvM) is a mathematical method heavily used in many programs and computations. In SMvM, values from an input vector are multiplied with values from a matrix and added into an output vector.

The aim of the thesis is to develop new ideas and algorithms to speed-up parallel SMvM on a shared memory computer.

Fundamentals In the paper the author discussed parallel programming, OpenMP, sparse matrix, coordinate representation (COO), Compressed row storage (CRS), Compressed column storage (CCS) and Parallel SMvM with various techniques.

The coordinate representation of sparse matrices consists of three arrays to store row indices, column indices and values of non-zero elements. The elements are easy to sort, restructure and allocate to a thread because the elements save its coordinate. CRS and CCS also save only the non zero elements, but they reduced the space used to store the coordinates. CRS reduced on row, and CCS reduced on column indices. For each row or column, the algorithms iterates over the elements of that row or column and multiplies it with the corresponding value from the input vector and adds the result into its correct place in the output vector.

After introduction part Mr. Tessem briefs about challenges for SMvM and discusses different existing solutions and their weaknesses.

Even work load The author researches matrix vector multiplication efficiency using various matrix storage representations and work distribution techniques (static, dynamic, guided, binary search). Inspired by min-makespan problem, the author reports some performance improvement of matrix-vector multiplication of CRS representation with binary search distribution. On arrowhead matrices with COO representation the work load distributes perfectly even. Therefore for parallel SMvM algorithms to have an even work load the COO-algorithm may be the better choice than CRS or CCS, even though it uses more memory.

Efficient cache use For the efficient cache using a good idea can be to use CRS-CSS-hybrid algorithm. But in some cases it can slow down the performance because there is a higher amount of data movement. Therefore efficiency of the hybrid- and CRS-algorithms depends on the structure of the matrix. The author poses poses that hybrid algorithms can be very efficient (more cache-efficient) on very sparse matrices or on random matrices.

Improving CSS with colouring The problem of colouring algorithms for efficient parallel steps is in turn lead to write conflicts and use of the locks, potentially reducing efficiency and speed-up. For solving this problem author suggested to make the colouring random with the binary search. However the algorithm is using longer running time with larger distance between columns that gets the same colour. When the number of columns using each colour is high, a random colouring algorithm shows some speed-up because it makes each parallel step equally efficient, but is impractical due colouring time consumption.

Distribution	p=1	p=2	p=4	p=8	p=16	p=32	p=64
machine 1: static, coo	0.334	0.249	0.228	0.24	0.241	0.625	0.711
machine 2: static, coo	0.388	0.275	0.259	0.521	1.038	1.204	1.378

Table 1: Selected results on the COO-algorithm for delaynay_n24 matrix

Distribution	p=1	p=2	p=4	p=8	p=16	p=32	p=64
machine 1: static, crs	1.47	0.857	0.604	0.335	0.285	0.257	0.274
machine 1: dynamic, crs	4.635	7.146	6.298	4.687	2.252	2.247	2.243
machine 1: guided, crs	1.596	0.831	0.427	0.225	0.181	0.19	0.195
machine 2: static, crs	1.767	1.18	0.611	0.566	0.58	0.601	0.663
machine 2: dynamic, crs	4.756	7.269	6.179	5.181	5.108	5.006	4.954
machine 2: guided, crs	1.666	0.857	0.455	0.391	0.377	0.362	0.359

Table 2: Selected results on the CRS-algorithm with different distribution schemes for HV15R matrix.

2 Literature review assessment

The author provides a comprehensive review of previous approaches for improving the efficiency of cache use in parallel sparse matrix vector multiplication, avoiding write conflicts, and modifying for false sharing reduction.

However, in the discussion section of even the work load experiment, the report jumps right into the results without providing an understanding of how the approximation CRS-algorithm using binary search on the min-makespan problem works and why it gives better results on some matrices. This is not a problem for those who are familiar with the min-makespan problem, but others may be interested in reading about the approach's fundamentals.

3 Results and discussion

The experiments have shown that arrowhead matrices like delaunay_n24 in CRS format do not converge due huge difference in work load between heaviest and lightest loaded thread for static, dynamic or guided distributions. However they do converge in COO format because there is no difference MPI distributes the work load evenly in COO format (Table 1).

For HV15R matrix there are some acceleration (up to 17%) in original experiments for binary search scheduling on 2-8 threads. In our implementation the biggest difference (about 29.5%) in results is on 2 thread experiment between static distribution of second machine and guided distribution on first machine Table 2.

For nlpkkt240 and af_shell10 matrices static scheduling is better than guided in 1-4 thread parallelization on both machines, but starting with 16 thread parallelization guided scheduling has some performance advantages on second machine Table 4, Table 3. In the original implementation there are no differences between all work distributions except dynamic.

We can confirm that the dynamic distribution performs the worst for ordinal sparse matrices in comparison to guided, static and binary search scheduling techniques. The author's machine most probably has more than 16 cores because there are still improvements after parallelization to 8 processors and more. In our replication there are always no improvement or degradation in performance on second machine in parallelization on 16 and more threads. Most likely because the second machine has only 8 cores Table 5.

Distribution	p=1	p=2	p=4	p=8	p=16	p=32	p=64
machine 1: static, crs	0.52	0.311	0.169	0.124	0.108	0.104	0.104
machine 1: dynamic, crs	0.936	0.898	0.67	0.498	0.267	0.281	0.272
machine 1: guided, crs	0.68	0.344	0.177	0.098	0.091	0.081	0.077
machine 2: static, crs	0.514	0.295	0.161	0.131	0.154	0.157	0.172
machine 2: dynamic, crs	0.885	0.786	0.66	0.511	0.518	0.513	0.508
machine 2: guided, crs	0.612	0.325	0.168	0.112	0.117	0.096	0.095

Table 3: Selected results on the CRS-algorithm with different distribution schemes for af_shell10 matrix

Distribution	p=1	p=2	p=4	p=8	p=16	p=32	p=64
machine 1: dynamic, crs	15.595	11.596	10.91	7.581	3.933	3.929	3.941
machine 1: guided, crs	11.175	6.43	3.257	1.665	1.216	1.238	1.173
machine 2: static, crs	8.844	5.498	3.348	1.806	2.272	2.852	3.537
machine 2: dynamic, crs	15.047	12.261	10.406	7.691	7.677	7.58	7.56
machine 2: guided, crs	10.867	6.271	3.198	1.628	1.622	1.587	1.657

Table 4: Selected results on the CRS-algorithm with different distribution schemes for nlpkkt240 matrix

Machine	Memory(RAM)	Video	CPU	number of cores
machine 1	2 x 16 GB DDR4	NVIDIA GeForce RTX 3080 (10 GB)	OctalCore Intel Core i7-10700F, 4600 MHz	16
machine 2	16 GB DDR4	NVIDIA GeForce RTX 2080 SUPER (8 GB)	OctalCore Intel Core i7-9700K, 3600 MHz	8

Table 5: Characteristics of replication machines