Performance of Sparse Matrix-Multiple Vectors Multiplication on Multicore and GPUs

Walid Abu-Sufah
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
and
The University of Jordan
abusufah@illinois.edu

Khalid Ahmad
Computer Engineering Department
University of Jordan
Amman, Jordan
Khalid.ahmad@computer.org

Extended Abstract

Sparse matrix-vector multiplication (SpMV) is a performance bottleneck in iterative methods for solving large sparse linear systems and eigenvalue problems. We implemented a heuristics-based auto-tuning framework for SpMV on NVIDIA GPUs. For a given sparse matrix, our auto-tuner delivers the highest performing SpMV kernel which combines the use of the most efficient storage format and tuned parameters of the corresponding CUDA code targeting the underlying GPU architecture [1][2]. So far, our auto-tuner has dealt with the following sparse matrix storage schemes: Diagonal (DIA), ELLPACK (ELL), CSR (vector), Coordinate (COO), Hybrid (HYB) [3], BELLPACK [4], and our own BTJAD [5].

Currently, our interest is focused on the integration of yet another important type of sparse matrix kernels in our auto-tuner, sparse matrix-multiple vector multiplication (SpMM) kernels. SpMM is a generalization of SpMV in which a sparse n-by-m matrix A is multiplied by a tall and narrow dense n-by-k matrix B (k<<n). SpMM is used in a variety of computations such as those using block Krylov subspace methods for improving the performance on parallel architectures through replacing SpMV kernels by their SpMM counterparts [6].

Different storage schemes/kernels work best for matrices with different sparsity patterns. When nonzero values are restricted to a small number of matrix diagonals, the diagonal storage format (DIA) is the most efficient. In this poster, we refer to such matrices as structured matrices. DIA efficiently encodes matrices arising from the application of stencils to regular grids, a common discretization method. For structured matrices, DIA-SpMV is the best performing SpMV kernel [1][3][5]. For matrices with uniform row lengths (i.e. the maximum number of nonzeros per row does not substantially deviate from the average), the ELLPACK/ITPACK storage format (ELL) is the most efficient. This is often the case with matrices obtained from semi-structured meshes and well-behaved unstructured meshes [3]. For matrices with uniform row lengths, ELL-SpMV is the best performing SpMV kernel [1][3][5].

So far, we have designed and implemented two CUDA 6 SpMM kernels targeted at structured matrices (DIA-SpMM) and matrices with uniform row lengths (ELL-SpMM). Work is continuing on the design, implementation, auto-tuning, and performance evaluation of other SpMM kernels which target matrices with other types of sparsity patterns. In this poster we compare the performance of our kernels with SpMV kernels in NVIDIA CUSP and CuSPARSE libraries. Moreover, we do a comparison with the performance of the SpMM kernel available in CuSPARSE. We also measure how our kernels executing on a GPU compare with the performance of SpMV kernels of Intel MKL library executing on recent multi core processors. Our kernels have superior performance because; (i) we use the most efficient storage schemes for the test matrices, (ii) we multiply each matrix by a block of vectors instead of one, and (iii) our code makes the most of GPU registers to exploit data reuse in SpMM. The design skeletons of our kernels are described in [7].

We performed our measurements using two nodes of two different HPC clusters. On the NVIDIA benchmarking PSG cluster node we executed the kernels on state-of-the-art Kepler K40m GPU. The Intel MKL SpMV kernel was executed on a dual socket 10-core Intel Ivy Bridge E5-2690 v2 @ 3.00GHz CPU. The number of CPU threads was varied between 1 and 20. On the Cyprus Institute Cy-Tera cluster node we used the Fermi M2070 GPU and a dual socket Intel Westmere 6-core X5650 @ 2.66GHz CPU. The number of CPU threads was varied between 1 and 12. Performance is evaluated in terms of GFLOP/s, which is computed by dividing the number of arithmetic operations by the average running time of 1000 runs. Our time measurements do not include the time required to transfer data between the host and the GPU. Our experiments test our SpMM implementations for multiplying a sparse matrix by up to 512 vectors.

For our measurements, we used 5 structured grid matrices that represent common stencil operations on regular grids (Table 1) [3]. They are formed by applying a Laplace stencil operator to every point in an N-dimensional space (i.e. 1, 2, and 3 dimensions). The number of diagonals is the number of points in the stencil. This results in nonzero elements that are restricted to a small number of diagonals, where the number of diagonals is the number of points in the stencil. From the

University of Florida collection and from 19 application areas we used another 28 structured matrices (Table 2) and 29 matrices with uniform row lengths (Table 3).The details of the characteristics of all matrices we used can be found on our project website [8]

TABLE I
LAPLACIAN OPERATORS DISCRETIZED AS K-POINT FINITE DIFFERENCE STENCILS ON REGULAR GRIDS.

Matrix	Grid	Diagonals	Nonzero Elements
Laplace 3pt	1,000,000	3	2,999,997
Laplace 5pt	$(1,000)^2$	5	4,994,001
Laplace 7pt	$(100)^3$	7	6,626,349
Laplace 9pt	$(1,000)^2$	9	8,986,005
Laplace 27pt	$(100)^3$	27	26,178,647

TABLE II
28 UNIVERSITY OF FLORIDA STRUCTURED MATRICES

Matrix	Nonzeros	Diagonals
Maximum	10,319,760	99
Average	1,537,409	16.6
Minimum	19,996	5

TABLE III
29 UNIVERSITY OF FLORIDA UNIFORM ROW LENGTH MATRICES

Matrix	Nonzeros	Average nonzeros per row	Maximum nonzeros per row	Standard deviation
Maximum	7,791,168	72.1	81	19.1
Average	1,339,107	10.3	10. 9	1.1
Minimum	20,360	1	1	0

SpMV kernels in the highly optimized CuSPARSE library do not support the DIA or the ELL storage formats. Hence, for structured matrices, we compare the performance of our DIA-SpMM with CUSP DIA-SpMV and for matrices with uniform row lengths we compare our ELL-SpMM with CUSP ELL-SpMV. CuSPARSE provides a SpMM implementation for the widely used compressed sparse row format (CSR). Thus, we compare the performance of our SpMM kernels to the CuSPARSE CSR-SpMM. Using the 5 Laplace matrices, we compare the performance of our DIA-SpMM kernels on the K40m GPU to the performance of Intel MKL CSR-SpMV on the Ivy Bridge and Westmere CPUs.

We note that our SpMM kernels achieve significant speed improvements over NVIDIA's and Intel's SpMV library kernels. On the Kepler K40m: (i) for structured matrices the *average* speed improvement of our DIA-SpMM over CUSP DIA-SpMV is 2.4x, and the *maximum* is 4.8x, and (ii) for matrices with uniform row lengths the *average* speed improvement of our ELL-SpMM over CUSP ELL-SpMV is 2.8x and the *maximum* is 4.5x

For structured matrices, our DIA-SpMM kernel is 5.2x faster than the CUSPARSE CSR-SpMV. The maximum speedup is 6.5x. For matrices with uniform row lengths, our ELL-SpMM is 3.9x faster than CUSPARSE CSR-SpMV. The maximum speedup is 8.3x

For Structured matrices, our DIA-SpMM kernel is 2x faster than CUSPARSE SpMM. The maximum speedup is 2.5x. For matrices with uniform row lengths, our ELL-SpMM is 1.6x faster than CUSPARSE SpMM. The maximum speedup is 2.8x.

For each of the 5 Laplace matrices, different numbers of threads are used by MKL 11.0 CSR-SpMV to produce the highest performance. This is the case on both the Intel Ivy Bridge dual socket 10-core E5-2690 v2 and the dual socket Intel Westmere 6-core X5650. On the Ivy Bridge, the *average* speedup of our DIA-SpMM on the K40m over the best performing MKL SpMV kernel is 7.2x. The maximum speedup is 12.3x (MKL SpMV using 9 threads to achieve the best performance for the Laplace 27pt matrix). On the Westmere, the *average* speedup is 22.8x. The maximum speedup is 27.5x (MKL SpMV using 11 threads to achieve the best performance for the Laplace 9pt matrix).

The improvements achieved by your kernels executing on the Fermi M2070 are close to the improvements achieved on the Kepler K40m.

REFERENCES

[1] W. Abu-Sufah and A. Abdel-Karim, "Auto-tuning of Sparse Matrix-Vector Multiplication on Graphics Processors", Proceedings of the International Supercomputing Conference (ISC'13), Leipzig, Germany, June 16-20, 2013, J.M. Kunkel, T. Ludwig, and H. Meuer, eds., vol. 7905 of Lecture Notes in Computer Science, Springer, Berlin / Heidelberg, 2013, pp. 151-164.

[2] W. Abu-Sufah, "A Library of Automatically Optimized Sparse Matrix Computations Kernels on Graphics Processors Accelerators", [Retrieved July, 2014]. [Online]. Available: https://sites.google.com/site/sparseautotuner/

[3] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. ACM, SC 2009, p. 1-11.

[4] Choi J., Singh A. and Vuduc R. , Model-driven Autotuning of Sparse Matrix-vector Multiply on GPUs", proceedings of PPoPP '10, Bangalore, India, January 9–14, p. 37–48.

[5] W. Abu-Sufah and A. Abdel-Karim, "An Effective Approach for Implementing Sparse Matrix-Vector Multiplication on Graphics Processing Units", Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications (HPCC-2012), Liverpool, UK, June 2012, pp. 453-460, DOI 10.1109/HPCC.2012.68.

[6] H. M. Aktulga, A. Buluc, S. Williams, C. Yang, "Optimizing Sparse Matrix-Multiple Vector Multiplication for Nuclear Configuration Interaction Calculations", 2014 International Parallel and Distributed Processing Symposium (IPDPS 2014), May 2014.

[7] W. Abu-Sufah and K. Ahmad, "On Implementing Sparse Matrix Multi-Vector Multiplication on GPUs", Proceedings of the Workshop on GPU Computing, held in conjunction with the 16th IEEE International Conference on High Performance Computing and Communications (HPCC-2014), Paris, France, August 20-22, 2014.

[8] W. Abu-Sufah, "Matrices used in Performance of Sparse Matrix-Multiple Vectors Multiplication on Multicore and GPUs", [Retrieved July, 2014]. [Online]. Available: https://sites.google.com/site/sparseautotuner/matrices/performance-of-sparse-matrix-multiple-vectors-multiplication-on-multicore-and-gpus