Optimized Sparse Matrix Formats on GT200 and Fermi GPUs

List of authors:

M. Hugues [1] and S. Petiton

CNRS/LIFL, Laboratoire d'Informatique Fondamentale de Lille, France

maxime.hugues@lifl.fr, serge.petiton@lifl.fr [2]

The rise of computing power now goes through the multiplication of cores on a single-chip because of the frequencies limitation. Multicore processors will compose future peta and exascale supercomputers which will allow applications to sustain many petaflops and thus to make breakthrough on scientific challenge problems. One massively multicore processor already available on the market is the GPU and has been making the rebirth of data parallel programming which is focused on data mapping and movements. GPU is a high peak performance architecture which gives the opportunity to accelerate scientific applications. Dense linear algebra has already shown that it could benefit from this architecture [1] and some libraries such as CUFFT or CUBLAS are available to easily develop high performance applications. In opposite, sparse matrices have irregular data structures which generate a memory bottleneck by accessing data in a non-contiguous fashion. The main optimization strategy is to get the most efficient memory accesses by arranging data in a well-suited manner depending on the matrix structure. Sparse formats define this data arrangement in memory. The sparse matrix vector product (SpMV or $Ax = y$) is one of the most important kernel in scientific computing. This kernel is intensively used and is the main performance key of iterative methods, such as Conjugate Gradient or GMRES, for solving partial differential equations, large linear systems or eigenvalues problems. Thereby, many researches have been conducted to optimize this kernel implementation through various sparse formats on many-core processors. On multicore processors, the Block Compressed Sparse Row (BCSR) format has been evaluated and optimized on different platforms like Cell BE, Intel, AMD and Sun processors [2]. It clearly achieves the best performances by reusing sparse matrix elements in the different cache levels thanks to the register and cache blocking techniques. On massively parallel architectures, some implementations of SpMV kernel have been evaluated by using various sparse formats on NVIDIA GPUs [3]. They have shown that their HYB Format, a mix between the COO and Ellpack/ITpack format, achieves the best results on a set of matrices previously used on multicore processors and the DIA and Ellpack/ITpack is the most efficient format for a finite differences discretization of the Laplacian. In [4], authors have also proposed an optimized implementation of the Compressed Sparse Row (CSR) format by using vectorization and padding which increase the efficiency and outperform the implementation of Nvidia. A sparse linear solver [5] method has been developed on GPU and has proposed the implementation of the SpMV by using the BCSR format for register blocking. In spite the fact, GPU is called data parallel architecture, like Connection Machine, the past results obtained on this type of architecture are not fully reusable as is.

[1]

[2]

We propose in this talk to study some implementations on GPU of sparse format which has been successful on data parallel machine (CSC, SGP [6]) and present a performance comparison of common sparse formats for the SpMV kernel with our optimizations. This work also compares the performance achievement on the GT200 and the new Nvidia architecture, Fermi, through two evaluation methodologies The first method is based on a set of matrices from the University of Florida Sparse Matrix Collection and a finite difference discretization of the Laplace operator in 2D with different stencil in order to compare our results to previous of [3]. The second one is focused on the sparse matrix format performances following the distribution of the non zeros values in the matrix by using a matrix pattern, named C-Diagonal q-perturbed. These patterns was introduced to evaluate sparse format performances on data parallel supercomputers following communications which depends on the distribution of nnz [7]. The C-Diagonal matrix is defined as a matrix with nc diagonals from the main diagonal to the right. The C-Diagonal q-perturbed matrix is built from a C-Diagonal matrix by modifying the data structure with a random perturbation. Each element of the matrix has a perturbation called q which defines whether the value has to be set at a random distance from the main diagonal, thus the element column index is modified.

On the set of matrices and the GT200, we demonstrate Ellpack/ITpack format with a column-major order in memory is the best format for common matrices by achieving 18.7Gflop/s in single precision and 11Gflop/s in double precision. Our variant of Ellpack/ITpack with a row-major order performs also well, depending on the matrix structure. For matrices with a nnz per row lower than 16, this implementation and all vectorized formats, such as the CSR implementation, perform poorly because they can not realize a coalescing with a half-warp. The proposed vectorized version of BCSR format achieves also good performance on matrices with a sufficient nnz to have an optimized blocking. Results on compressed column formats such as CSC, Ellpack/ITpack with column compression and SGP show that type of format perform less better than compressed row format. This comes from the summing of all elements belonging to the same row which makes irregular data access and degrades performances. The performance study of sparse formats following the distribution of nnz also shows interesting results about the choice of format for different matrix structure. Ellpack/ITpack implementation with column-major order stays the most efficient sparse format by performing 21Gflop/s in single precision and 13.8Gflop/s in double precision with $q = 0$. Nevertheless, we have seen for sparse matrices with a strong distribution of nnz ($q > 0.5$) that the various formats achieve approximately the same performance by reaching 2.0 Gflop/s in single and double precision. This can be interesting for iterative methods which require high precision to keep stability and converge faster without loosing performance. On Fermi, the performance behavior observed is the same with an improvement for some sparse formats such as CSR and BCSR. The first one benefits of a 1.7 SpeedUp in comparison to the GT200 by achieving 17Gflop/s and the second one realizes 25Gflop/s with a SpeedUp of 2. BCSR outperform Ellpack/ITpack format on Fermi in single and double precision by using the new cache levels on the GPU. The C-diagonal q-perturbed benchmark shows that Ellpack/ITpack is still the best because as the distribution of nnz grows the number of Block for BCSR also grows and creates extra zeros to compute. Moreover, we can notice the performance are quiet the same but they decline faster when q is growing than

on the GT200 architecture and finally reach less figures for $q = 1.0$ in single precision. In double precision, we observe a SpeedUp of 1.5 when the distribution of nnz increases and demonstrates the effort of Nvidia on Fermi to optimize double precision computations is clearly significant.

References

[1] V. Volkov and J. Demmel, Benchmarking GPUs to Tune Dense Linear Algebra, *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 31, 2008.

[2] S. Williams, L. Oliker, R. W. Vuduc, J. Shalf, K. A. Yelick and J. Demmel, Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms, *SC'07: Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing*, 38, 2007.

[3] N. Bell and M. Garland, Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors, *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 1–11, 2009.

[4] M. Baskaran and R. Bordawekar, Optimizing Sparse Matrix-Vector Multiplication on GPUs. *IBM Technical Report*, 2009.

[5] L. Buatois, G. Caumon and B. Levy, Concurrent Number Cruncher: a GPU Implementation of a General Sparse Linear Solver, *Int. J. Parallel Emerg. Distrib. Syst.*, 24:205–223, 2009.

[6] W. R. Ferng, S. G. Petiton and Kesheng Wu, Basic Sparse Computations on Data Parallel Computers, *PPSC*, 462–466, 1993.

[7] S. G. Petiton and C. Weill-Duflo, Massively Parallel Preconditioners for the Sparse Conjugate Gradient Method, *Parallel Processing: CONPAR 92 - VAPP V, Second Joint International Conference on Vector and Parallel Processing*, 373-378, 1992.