Architecture Independent Performance Evaluation of Sparse Matrix-Vector Multiplication on Multi-Core Processors

Hussein Moghnieh and David A. Lowther
Department of Electrical and Computer Engineering, McGill University
3480 University Street, Montreal, Quebec, H3A 2A7, Canada
hussein.moghnieh@mail.mcgill.ca, david.lowther@mcgill.ca

Abstract — Sparse matrix-vector multiplication (SpMV) is an essential kernel used in many scientific applications and although it is parallelizable, it only reaches a small amount of the attainable peak performance of the processor. As the core count increases on a processor, using all cores to speedup SpMV can be a major resources waste. This paper attempts to understand the limitations of this kernel through quantifying its performance and relating it to the processor's parameters. This provides the basis for an appropriate resource mapping on multi-core processors.

I. INTRODUCTION

The SpMV is an essential kernel present in many numerical computations. In particular, it is important for the computational electromagnetics community, since it is an essential building block of conjugate gradient (CG) methods, which are used to solve a system of linear equations.

It is well known that matrix-vector multiplication ($y = A * x$) exhibits a low floating point operations count to memory access ratio. In this paper, we attempt to evaluate the performance of SpMV on any architecture, by relating the hardware architecture parameters to the matrix properties. Our first goal is to determine the optimal number of threads to be used for a specific problem on a specific processor. The second goal involves evaluating the performance of SpMV in the presence of other running methods and kernels on the remaining cores. These two goals are based on our argument that, as the number of cores increases, leveraging the power of a multi-core processor would be a combination of running fine grained parallel algorithms and other methods in an ad-hoc manner. It is crucial to understand each kernel and to be able to form a prediction of its behavior in the presence of other running methods.

II. OVERVIEW

Sparse matrix-vector multiplication is ubiquitous in many scientific domains. There is a wide literature attempting to optimize this kernel to a specific computer architecture. The interest of this paper resides in running such a kernel on a multi-core based processor.

It is clear that matrix-vector multiplication, whether the matrix is dense or sparse, is parallelizable, since many operations can be performed simultaneously. Nevertheless, the low attainable FLOPS is mainly due to a memory bottleneck. The matrix A presents a problem due to the large number of memory operations with no data reuse.

The peak performance attainable in sparse matrix-vector multiplication is always less than that of a dense matrix of similar size. This is mainly due to the following: First, a more complex data storage structure is needed in order to store the non-zeros, an example is Compressed Row Storage (CRS)[1]; this will lead to indirect access to the matrix A. Second, the non-zero pattern is unpredictable, which could cause unpredictable access to the x and y vectors. Finally, for some matrices with low non-zero count per row, the loop setup overhead would dominate the time of the calculation and would not be able to be amortized over the short calculation time of a few non-zeros. The author in [2]has examined the extent of these effects and found that the problems mentioned above vary greatly, depending upon the problem and the processor's architecture.

Further attempts to understand and optimize the performance of the SpMV kernel on multi-core based processors have been described in [3-4]. They have identified many serial (e.g. register blocking, cache blocking, etc.) optimization strategies and parallel decomposition techniques and applied them on a set of test matrices and different multi-core processor architectures. The results have shown that the effect of an optimization technique depends on the matrix and the platform. An investigation of this variation has been carried out qualitatively, that is by observing the particularity of those matrices which provided high performance, low performance, or discrepancy of performance amongst different architectures.

Despite the fact that the above mentioned techniques can provide a performance gain, SpMV is still reaching only a low percentage of the processor's theoretical peak performance; for an Intel Clovertown processor, the attainable peak performance has been less than 5% of the theoretical peak[3]on average.

III. METHODOLOGY

This paper attempts to obtain a quantitative estimation of the performance of SpMV (serial and parallel) and to further understand how this performance is altered when SpMV runs as part of a CG solver, or when multiple SpMV's run simultaneously. As the number of processor cores increases on a desktop computer, using the cores for multiple tasks, as opposed to using all the cores for a specific parallel kernel, would require resource mapping and is a significant challenge.

In the first set of experiments, the effect of cache misses of the vector x will be assessed in a way similar to that performed in [2]. An efficient, non-optimized implementation of SpMV will be compared to another version which does not multiply by x, but rather by a fixed value, thus removing the cache misses. Of course, the result of the latter routine is not correct. The runtime ratio has been obtained for experimental runs using a set of matrices as shown in Fig.1(Experimental). This ratio shows the magnitude of the effect of the cache misses on x.

In order to analytically understand such an effect, a distance model has been developed that takes into account the cache line size of the processor used (Intel Quad core Q6600-8 MB L2 - 2.4 GHz). The vector which contains the column CRS format (which contains the column x indices), is analyzed to determine where each jump in the array that is larger than the cache line size will occur and this will count as a cache miss. The total number of cache misses are obtained and divided by the number of the non-zeros of the matrix. The result is plotted in Fig.1Fig.1(Analytical). This figure shows that the magnitude of the effect of cache misses can be predicted by looking at the access pattern of the $x$vector.

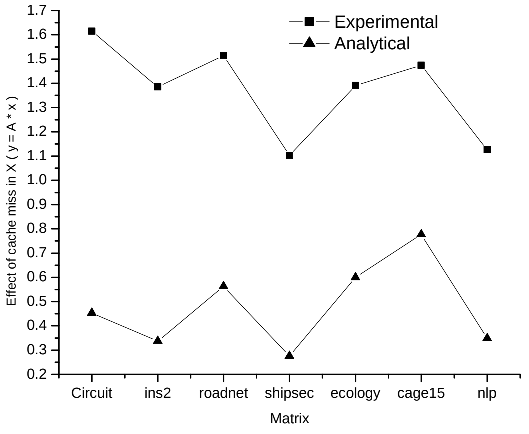

Fig.1. Cache miss effect using the distance model

IV. PARALLEL SPMV

A parallel SpMV has been applied to two sets of matrices obtained from [5]. The first set has a storage size in CRS format ranging between 10 MB and 100MB (Fig.2), the latter have a storage size in CRS format larger than 500MB (Fig.3). In the parallel SPVM used, the matrix is divided into blocks of rows where each block is assigned to a thread. By using this non-optimized version of parallel SpMV, the magnitude of the effect of cache misses could be related to the matrix's and processor's properties in order to know which optimization techniques to use and what number of threads to utilize.

The preliminary results show that for matrices less than 100MB, a speedup gain could be obtained for a number of threads larger than the number of available cores (although the processor used does not support hyper-threading), while this was not true for large matrices where using 2 threads was the optimal choice.

V. CONCLUSION

The SpMV exhibits different performance behavior as the number of threads increases. By using a non-optimized parallel SpMV and analyzing the memory access patterns, the effect of the available tuning techniques can be predicted. This will be the main work of the extended version of this paper, i.e. to understand the limitation of SpMV and the optimal number of threads to use. This information will form the basis of algorithms and allocation techniques which will lead to a more effective utilization of the multi-core processor.

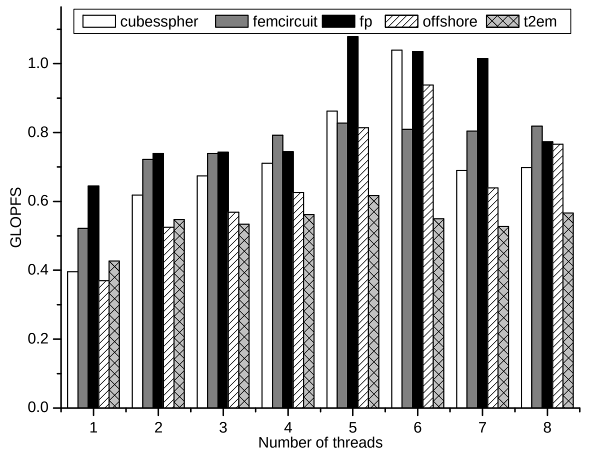

Fig.2. Matrices from electromagnetics problems (CRS storage < 100MB)

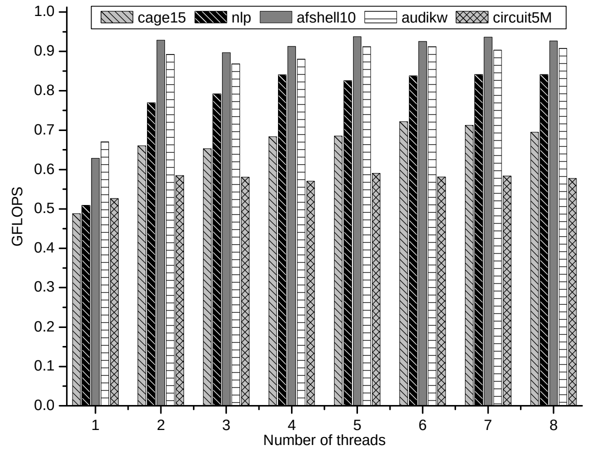

Fig.3. Miscellaneous Matrices (CRS storage > 500 MB)

VI. REFERENCES

[1] R. Barrett, "Templates for the solution of linear systems: building blocks for iterative methods," ed: Society for Industrial Mathematics, 1994, pp. 63-65.

[2] G. Goumas, *et al.*, "Understanding the performance of sparse matrix-vector multiplication," in *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, PDP 2008*, 2008, pp. 283-292.

[3] S. Williams, *et al.*, "Optimization of sparse matrix–vector multiplication on emerging multicore platforms," *Parallel Computing,* vol. 35, pp. 178-194, 2009.

[4] R. Vuduc, *et al.*, "OSKI: A library of automatically tuned sparse matrix kernels," *Journal of Physics: Conference Series,* vol. 16, pp. 521-530, 2005.

[5] University of Florida sparse matrix collection , December 2010 [Online]. Available: www.cise.ufl.edu/research/sparse/matrices