Scalability of Hybrid Sparse Matrix Dense Vector (SpMV) Multiplication

Brian A. Page
CSE Dept.
Univ. of Notre Dame
Notre Dame, IN USA
bpage1@nd.edu

***Abstract*—TBD**

***Keywords*-Scalability, Hybrid SpMV;**

I. Introduction

II. Related Work

Due to the computation impact that SpMV operations have on a many scientific applications there has been an effort to analyze its performance and scalablity characteristics. Bylina, Bylina, Stpiczunski, and Szalkowski [1] introduced and evaluated the performance of both multicore and multinodal implementations of SpMV on various chip architectures. A modified version of the SpMV algorithm found in the SPARSKIT Fortran library [2] for the multicore implementation. Using matrices from the University of Florida Sparse Matrix Collection (UFSMC), they found that for their multicore algoritm, similar performance was experienced accross all matrices tested when the number of threads remained low. Alternatively as architectures allow for increased thread count, higher performance can be obtained, and it was noted that the use of OpenMP allowed for performance comperable to that of the optimized Intel MKL version of SpMV cite Intel MKL ?.

Bylina et al's multinodal implementation distributed equal sized sub matrices of a given benchmark matrix to each MPI process, where in each process would then work on the non-zeros contained within that submatrix via a multithreaded version of Intel's MKL SpMV routines. For the distribution method chosen the density in addition to distribution of non-zeros within the matrix had the greatest impact on the scalability of their multinodal algorithm.

-discuss memory bandwidth as it impacts SpMV

III. Implementation

The application that was written to emulate the behavior of that used in the study performed by Bylina et al was written using the C++ programming language. We chose to forgo the use of proprietary libraries such as BLACS and MKL deciding instead to write explicit MPI and OpenMP directives to control distributed and shared memory behavior across the cluster environment. Special care was paid to insure that the communication pattern matched that of the 2d cluster methods in the BLACS library which were used in the prior work. It was felt that by not including these packages greater control over communication and memory access parameters could be achieved, even though the result may not be as highly optimized for particular applications, architectures, or compilers.

Benchmark matrices in the Matrix Market Format were chosen from the University of Florida Sparse Matrix Collection. The characteristics of the matrices chosen is discussed in greater detail in section 4a "Benchmarks". Matrices are read from file by the master MPI process and converted to Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC) format, depending on the individual matrix being input. Both CSR and CSC formats provide a reduction on memory requirements thereby increasing performance while reducing data transfer in multinode environments since only data about non-zero elements in the input matrix are kept this needs a cite. While the matrix is being read in, the distribution of work amongst the MPI processes is being determined. The distribution pattern used by Bylina et al and which we have emulated, we call the $submatrixmethod$, splits the input matrix A into p^2 sub matrices in which each piece has nearly identical dimensions based on the number of processes p and the row or column count of A. Therein the size of each sub matrix from A_p will be A_{rows}/p x A_{cols}/p.

Each MPI process will receive its assigned sub matrix in CSR/CSC format, which is stored as three vectors belonging to a csrSpMV class object, for storing row column and data values for each non-zero. On the master process a two dimensional vector of csrSpMV objects, the dimensions of which equate to the total number of MPI processes being used. For our purposes we chose to require that the number of MPI processes be a positive square value. This insures that the number of logical rows in the cluster environment is equivalent to the number of logical columns. It is important for this distribution method that this be the case, as MPI communication takes advantage of column masters and row masters for data distribution in addition to reduction and gather efficiencies.

As each new non-zero is read the sub matrix it is to be assigned to is easily determined based off its row and column, with the

A. Work Distribution

IV. Evaluation

- cluster architecture, network connectivity, number of nodes (max used), etc. - how timings where taken and GFlops calculated based on these time measurements. - note clock precision (nanoseconds) - number of tests run per test permuation - best, worst, and averages were taken for all times recorded and GFlops calculated

- then tables and pretty graphs to talk about

A. Benchmarks

why Parabolic_Fem, bmw3_2, torso1, and nd24k were chosen. -sparsity/density -size -symmetry

- how the GFlops will be calculated for symmetric and non-symmectric matrices (they are the same, however the Bylinia paper did some weird shit with their numbers!)

B. Impact: Process Count

C. Impact: Thread Count

asdfsadf

D. Impact: Sparsity

V. Future Work

VI. Conclusions

Acknowledgment

References

[1] B. Bylina, J. Bylina, P. Stpiczyski, and D. Szakowski, "Performance analysis of multicore and multinodal implementation of spmv operation," *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, vol. 2, pp. 569–576, Oct 2014.

[2] Y. Saad, "Sparskit: A basic tool kit for sparse matrix computations," *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, vol. 2, May 1990.