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

- unlike Bylina et al's multinoda algorithm, which used BLACS to perform the data distribution amongst the cluster, as MKL to perfrom the multithreaded SpMV on each cluster node, a hybrid distrbuted SpMV was designed as a single stand alone entity. - control over all MPI and OpenMP operations, including thread and process affinity, resource overscheduling, etc. - does not require software licensing - can be tailored to cluster size and node hoardware architecture

- the application was written in c++, compiled with openmpi with passthrough to the g++ compiler - compiled with openmp - optimization level 3

- Careful attention was paid to the MPI communication structure such that it would emulate the behavior produced by the BLACS routines used by Bylina et al for work distribution amongst cluster nodes. - this is not optimized, however does allow for greater control in evaluating potential scalability bottlenecks stemming from communication overhead

- Briefly discuss program flow/operation - similar to Bylinia et al the cluster is split into a grid consisting of n x n nodes in thereby requiring n^2 number of nodes in the cluster to function properly. - distribution determined on master, then sent to cluster column masters, and finally column masters send to remaining column members. - once data has been recieved by invididual nodes, they perform a multithreaded SpMV algorithm which utilizes OpenMP for parallelization - each thread works on a block of rows from the submatrix portion assigned to the MPI process - all threads have same number of rows in their blocks, though not necessarily the same number of non-zeros to work on due to matrix distrubtion - vectors provide potential increase in locality of reference due to being allcated contiguous portions of memory you will likely need to cite this - once all nodes in a cluster row have completed computation of their assigned work, an MPI_Reduce is called and performs a summation with the results being saved into a vector at the cluster row master for each cluster row. - the 0th column of cluster nodes contains all cluster row masters including the Global Master. It is on this cluster column that MPI_Gather is called in order to collect the distributed results back at the global master process.

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.