## Parallelism of Sparse Matrix Operations

Written by Clayton Herbst (22245091)

Focus Questions:

* What operating system did you use in your submission?
* What sparse matrix operations did you use?
* For each matrix operation implemented
  + Description of the parallelism implemented
  + Some informal reasoning about expected run-time and scalability
  + Testing results
  + Comment on the performance observed

##### Background

Matrices are the foundational structure used to represent the data around us. The computer’s ability to naturally store and manipulate this information in contiguous memory locations provides exceptional computation benefits to otherwise complicated calculations. Matrix-like data structures closely represent the relationship between various datasets, often called ‘vectors’ or matrices. The terms ‘vector’ and matrix have slight difference however are interchangeable for the most part. Both of these structures help physicians & mathematicians represent the complex systems in our world through systems of linear equations and the operations on these linear equations. Matrix computation on computer systems are often used to give good approximations of complicated/computationally intensive calculations (MIT News). In this way matrices can be used to store and represent nearly all sets of data. Other examples of applications include complex data structures such as binary trees, which can use matrix representation effectively to represent the relationship between various nodes within a tree (MIT News). Graphics rely heavily on matrices to represent pixels within a file and their respective RGB colours (MIT News). Graphical effects are the result of manipulating these pixel values using operations such as scalar matrix multiplication and matrix vector multiplication to create distortion. An understanding of the many implementations of matrices allows one to gain a greater appreciation for their importance in the modern data driven world. Any improvements gained on the computational efficiency of matrix operations are thus highly valuable to society and the focus of this report.

Through the implementation of parallel computing using the OpenMP library I have endeavoured to make use of modern computer architecture to gain efficiencies in the calculations of common matrix operations. Parallel computing … . The practical processing times of the parallelised matrix operations will be compared to the practical computation times of a traditional single threaded program. These results will be gathered using the macOS Mojave (v10.14.6) operating system, operating on a single Intel Core i5 processor with 2.3GHz processing speed and four physical cores. The hardware installed provides an L2 cache size of 256KB, L3 cache size of 6MB and 8GB of random-access memory (RAM). All results gathered and inferences made will be highly specific to the architecture used due to the nature of parallel computing and computing phenomena such as cache thrashing.

##### Sparse Matrix Data Structures

Sparse matrices are matrices where most of the elements are zeros. A matrix is generally considered sparse if less than 20% of its elements are non-zero. Sparse matrices introduce many complexities and inefficiencies into computer algorithms due to inflated time-complexities and space complexities.

Space complexity is the analysis of the relative amount of space required by a process relative to the size of its inputs. Storing large sparse matrices in their entirety is highly inefficient and often not feasible due to restrictions in hardware capabilities. Time complexity is the analysis of the relative time taken for a program/algorithm to execute relative to its input size. The time complexity of an algorithm and the space complexity required by the algorithm can be reduced by increasing the efficiency of subproblems within the algorithm. For sparse matrices, this can simply be achieved by compressing the sparse matrix into data structures that preserve the location of all elements however store the value of non-zero elements only. Thus, sub-operations can be performed on a reduced number of elements, improving the programs time complexity, while using less memory to store the information.

Typical sparse matrix data structures used to preserve the contents of sparse matrices are the Coordinate (COO) format, Compressed Row Storage (CRS) format and the Compressed Column Storage (CCS) format. CRS and CCS are very similar containing a subtle difference in how the order in which the non-zero elements are stored. It is thus assumed the characteristics of CRS and CCS with respect to storage and time complexities are very similar.

The Coordinate (COO) format sparse matrix data structure is the simplest compressed representation of sparse matrices. This data structure consists of three arrays the size of the number of non-zero elements. The non-zero element is stored in one array and the other two store the coordinates of the non-zero element. The space complexity of the COO data structure is , where n is the number of non-zero elements in the matrix. The coordinate format is advantageous when constructing sparse matrices, performing item-wise operations and allows for fast conversion into other sparse matrix data structures. However, improvements can be made on the storage complexity of the data structure.

The Compressed Row Storage (CRS) format and Compressed Column Storage (CCS) format provide increased non-zero element storage density. The space complexity required by the CRS data structure is , where *n* is the number of non-zero elements and *m* are the number of rows in the sparse matrix. This is achieved by storing all non-zero elements in a contiguous memory locations (), array referencing the number of non-zero elements in each row ( and an array referencing the column each non-zero element belongs to (). The trade-off of the reduced space-complexity is increased algorithm complexity due to the extra addressing step required for every scalar operation since the coordinates of the non-zero elements are not explicitly stored (Dongarra 1995). The time taken to construct the CCS data structure is also when reading from a dense matrix file format since elements cannot be read in a contiguous manner, rather need to jump ahead to fetch column elements and store elements in an erratic fashion. This data access pattern requires an interim COO data structure storage step and then reading into CCS data structure.

All three storage structures were implemented in my matrix operation command line tool in order to maximise the efficient and simple element-wise acccess provided by COO and make use of the unique data access patterns of CRS and CCS data structure to perform row-wise/column wise operations.

>>> Insert information about advantages and disadvantages of parallelising certain data structures.

##### Parallel Sparse Matrix Scalar Multiplication

Scalar multiplication involves the multiplication of a scalar number with each element of a given array. The time complexity of scalar multiplication on sparse matrices can be drastically improved by only performing the scalar multiplication on all non-zero elements. This simple reduction method exploits the fact that any element multiplied by zero equals zero. Hence all zero elements can be ignored by the algorithm, reducing the number of elements that need to be traversed to , where n is the number of non-zero elements. The complexity remains linear however in the context of sparse matrices this will be computationally noticeable as greater than 50% of elements are zero elements and hence ignored.

I have implemented the COO sparse matrix data structure in order to maximise the time the dense matrix file format can be read into the sparse matrix data structure as well as ensure the resultant matrix can be converted to the dense matrix file format efficiently.

##### Parallel Sparse Matrix Trace Calculation

##### Parallel Sparse Matrix Addition

##### Parallel Sparse Matrix Transposition

##### Parallel Sparse Matrix Vector Multiplication

##### References:

Hardesty L., 2013. *Explained: Matrices,* MIT News. http://news.mit.edu/2013/explained-

matrices-1206. [17 Sep 2019].

Dongarra J., 1995. *Compressed Row Storage. http://netlib.org/linalg/html\_templates*

*/node91.html. [22 Sep 2019].*