Joe Cavanagh was born in St. Paul, MN and grew up in nearby Woodbury. He currently attends the University of Minnesota–Morris, where he is pursuing a degree in computer science. Joe completed his first SULI appointment with the ASER group at Oak Ridge National Laboratory in August 2008. After graduation, he plans to work as a software engineer in the St. Paul area. In his spare time, Joe enjoys watching and playing basketball.

Xiaohui Cui is an associate research staff member in the Computational Sciences & Engineering Division of Oak Ridge National Laboratory. He received his Ph.D. degree in Computer Science and Engineering from University of Louisville in 2004. His research interests include swarm intelligence, high performance computing, agent based modeling and simulation, emergent behavior in complex systems, information retrieval and knowledge discovering. Dr. Cui has performed a number of research works in multi-agent systems, parallel and distributed knowledge discovering algorithms, and swarm-based social simulation. His current research focuses on developing new computational algorithms inspired from biological models.

Massively Parallel Latent Semantic Analyses Using a Graphics Processing Unit

Joseph Cavanagh and Xiaohui Cui

ABSTRACT

Latent Semantic Analysis (LSA) aims to reduce the dimensions of large term-document datasets using Singular Value Decomposition. However, with the ever-expanding size of datasets, current implementations are not fast enough to quickly and easily compute the results on a standard PC. A graphics processing unit (GPU) can solve some highly parallel problems much faster than a traditional sequential processor or central processing unit (CPU). Thus, a deployable system using a GPU to speed up large-scale LSA processes would be a much more effective choice (in terms of cost/performance ratio) than using a PC cluster. Due to the GPU's application-specific architecture, harnessing the GPU's computational prowess for LSA is a great challenge. We presented a parallel LSA implementation on the GPU, using NVIDIA® Compute Unified Device Architecture and Compute Unified Basic Linear Algebra Subprograms software. The performance of this implementation is compared to traditional LSA implementation on a CPU using an optimized Basic Linear Algebra Subprograms library. After implementation, we discovered that the GPU version of the algorithm was twice as fast for large matrices (1 000x1 000 and above) that had dimensions not divisible by 16. For large matrices that did have dimensions divisible by 16, the GPU algorithm ran five to six times faster than the CPU version. The large variation is due to architectural benefits of the GPU for matrices divisible by 16. It should be noted that the overall speeds for the CPU version did not vary from relative normal when the matrix dimensions were divisible by 16. Further research is needed in order to produce a fully implementable version of LSA. With that in mind, the research we presented shows that the GPU is a viable option for increasing the speed of LSA, in terms of cost/performance ratio.

Introduction

Considering the large amount of data being collected annually, methods are needed to extract valuable information from this data [1]. Latent Semantic Analysis (LSA) is a numerical technique used to extract information from large collections of text documents such as relationships between various terms, sentences and full documents [2]–[3]. LSA works by taking the singular value decomposition (SVD) of **A** where $\mathbf{A} = \mathbf{D}^*\mathbf{D}^T$, with **D** being the term-document matrix. A term-document matrix contains a numerical value for each term in its respective document, with the documents representing the columns of the matrix and the terms representing the rows of the matrix. Since there are often 5 000 or more unique terms, and documents may contain only a couple hundred terms, many of the matrix values become zero. The term-document matrix is created beforehand, using a term weighting and text stripping algorithm such as a Term Frequency-Inverse Document Frequency (TF IDF) [14]. The core of the SVD algorithm requires an eigendecomposition of the matrix, which has been a computational problem for many decades [2], [7]–[8], [11]. This makes SVD a computationally expensive algorithm, which makes it a prime candidate for decreasing processing times [2], [7], [11]. Many modern SVD methods tridiagonalize **A** before computing the SVD, due to the performance increase between calculating SVD on a normal matrix and a tridiagonal matrix [2], [7]–[8], [15]. A tridiagonal matrix is a matrix with values only in its main diagonal, one element above the main diagonal, and one element below the main diagonal. All other elements in the matrix are set to zero. Tridiagonalization takes up a large portion of the time for computing the SVD, so our main focus is our parallel algorithm which saves computation time.

Recently, the graphics processing unit (GPU) has become a focus for inexpensive, high performance computing in various scientific fields [10]. Due to the massively parallel architecture of the GPU, it is able to perform floating point calculations much faster than a standard central processing unit (CPU) [10]. We performed the CPU benchmarks on a Dell™ Precision 370 using SiSoftware® Sandra, a program for benchmarking various computer

components. When this paper was written, an Intel™ Pentium 4 3.6 GHz CPU was priced around $140 and produced 11 gigaflops (one billion floating point operations per second). An NVIDIA® 1 GB 8800 GT GPU cost around $200 and produced around 300 GFLOPS. The price-to-performance ratio for the CPU is 0.079 GFLOPS/dollar, while the GPU price-to-performance ratio is 1.5 GFLOPS/dollar. The impressive price-to-performance ratio of the GPU makes it a prime candidate for increasing the speed of LSA while still using components found in many desktop and laptop personal computers.

The increased performance of the GPU comes with certain difficulties. Memory transfer from system memory to GPU memory (host to device and device to host) remains relatively slow and can often be a bottleneck in many applications. Most current GPUs offer support for only single precision, while many scientific applications require double precision support. Also, certain algorithms are primarily serial, prohibiting much of the GPU's processing power from being utilized. For each of these problems a solution must be found. Currently, the effect of slow memory transfer can be minimized by performing as many GPU-based computations as possible between transfers. Transferring large amounts of data at one time is generally faster than making numerous small memory transfers. Double precision can be either emulated by an algorithm or can be obtained by purchasing a new NVIDIA® 200 series GPU.

Our literature research shows that there is currently no GPU-based implementation of LSA. Related research includes a team at the University of North Carolina at Chapel Hill that used a GPU-based algorithm for solving dense linear systems [4]. Their implementation of an LU decomposition algorithm performed 35% better than an Automatically Tuned Linear Algebra Subroutines (ATLAS) implementation on the CPU for matrices of rank 3500. Manavski and Valle have implemented the Smith-Waterman algorithm on the GPU [9]. The Smith-Waterman algorithm explores alignments between two sequences in protein and DNA databases. Their implementation ran between two and 30 times faster than other implementations for commodity hardware. Another example of the power of the GPU is given in the literature [13]. Robert, Schoepke and Bieri used the GPU to implement ray tracing algorithms and then compared the results to an implementation on a CPU. In an animated example, the GPU performed around six times better than the CPU. These examples show that the GPU is a powerful tool that, when used correctly, can be used to increase the speeds of a variety of algorithms in a variety of fields.

Materials and Methods

For our implementation, we decided to use a Lanczos algorithm to assist with the SVD. The Lanczos algorithm tridiagonalizes a matrix which allows for the computation of SVD to be performed significantly faster [8]. This is done by using various matrix-vector and vector-vector operations in order to achieve a Krylov subspace. This subspace is a representation of the original matrix and maintains approximate eigenvalues and eigenvectors to that of the original matrix. The reason we chose Lanczos is that it is not as computationally demanding as alternative options. The algorithm can be seen in Figure 1. The vector *alpha* is then used to form the main diagonal of the new matrix, where the vector *beta* will form the subdiagonal and superdiagonal. This algorithm is proven to be accurate in an environment without rounding errors. However, due to the fact that our GPU supports single precision, rounding errors are inevitable. Maintaining accuracy while using the GPU will be discussed more in the *Discussion and Conclusion* section.

Algorithm Lanczos
$v_1 \leftarrow$ random vector with norm 1.
$v_0 \leftarrow 0$
$\beta_1 \leftarrow 0$
Iteration: for $j = 1, 2, \cdots, m$
$w_j \leftarrow Av_j - \beta_j v_{j-1}$
$\alpha_j \leftarrow (w_j, v_j)$
$w_j \leftarrow w_j - \alpha_j v_j$
$\beta_{j+1} \leftarrow \|w_j\|$
$v_{j+1} \leftarrow w_j / \beta_{j+1}$
return

Figure 1. Lanczos Algorithm [8].

Our GPU implementation uses Compute Unified Basic Linear Algebra Subprograms (CUBLAS) to perform the matrix-vector and vector-vector operations that the algorithm requires. CUBLAS is a Compute Unified Device Architecture (CUDA) implementation of Basic Linear Algebra Subprograms (BLAS) which has been tuned to provide good performance across a variety of GPUs. To avoid bias, we compare our performance with the Intel® Math Kernel Library. The linear algebra routines from the BLAS libraries used in both the CPU and GPU implementations were identical. The only difference was that the algorithmic designs of each version were used in order to extract performance out of their respective architectures. The main linear algebra routines used are *sgemv*, *saxpy*, *sdot* and *snrm2*. These routines are frequently used for basic linear algebra functions that were developed to provide building blocks for larger applications. If $\mathbf{A} = \mathbf{D}^*\mathbf{D}^T$, with **D** representing the term-document matrix, only **A** needs to be stored in memory on the GPU, along with a few vectors which are a fraction of the data size of **D**. This is very advantageous, as the memory on a GPU card can be a very limiting factor. The card used for our testing has 1 GB of memory, which allows us to allocate about 950 MB to use in our program. The extra memory which cannot be allocated is due to a process behind the scenes preventing full allocation. The CUDA community currently regards this as a bug as it is unclear why this much memory is reserved for other uses. With 950 MB of usable memory, we are able to allocate matrices that exceed 15 000x15 000. This is well within the matrix size targeted for our algorithm.

After implementing the CPU- and GPU-based algorithm, we timed each implementation for various matrix sizes. The computer testing the implementations has the following specifications: one 3.6 GHz Intel Pentium 4 CPU, 3.00 GB of RAM, NVIDIA® 8 800 GT with 1 GB device memory, 160 GB hard drive. Three randomly generated matrix sizes were selected in an interval thought to clearly display the performance of each implementation, to perform the LSA of each matrix. For each of the three matrices, both CPU and GPU versions are run five times and the total time is averaged. This produces 15 total runs for both the CPU and

GPU at every matrix size interval. The times are then averaged for display purposes.

Results

Our initial results can be seen in Figure 2, which is for matrices up to 4000x4000. These initial results show a performance increase of two to six times. Figure 3 shows the average CPU and GPU run times for matrices that have dimensions divisible by 16. The tests were performed in the same manner as the original tests, with the only change being the matrix sizes. The GPU in this scenario is able to process the data from four times faster (for a 1 600x1 600 matrix) up to almost seven times faster (for a 5 600x5 600 matrix).

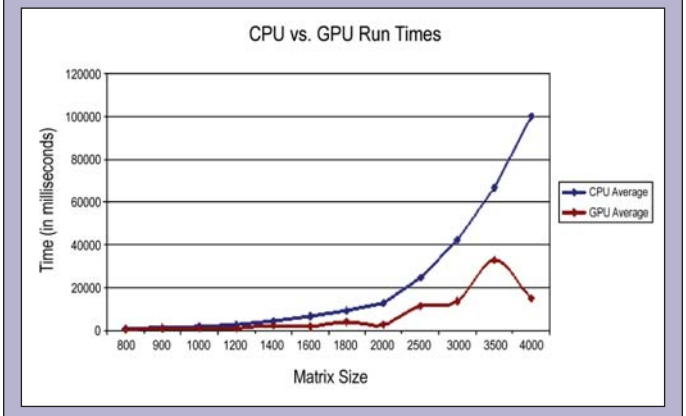

Figure 2. Shows test results for increasingly large matrices, up to 4 000x4 000.

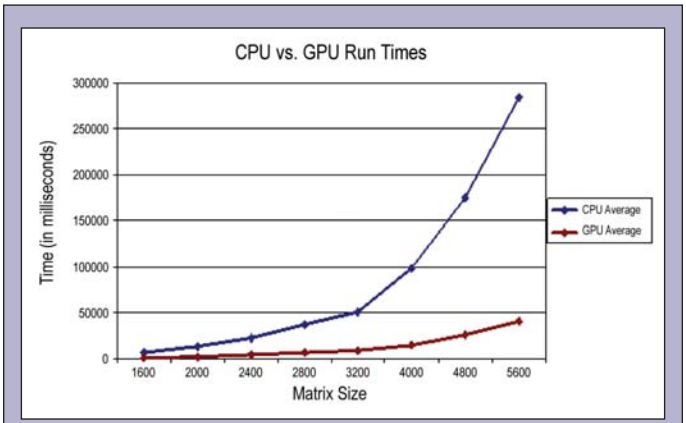

Figure 3. Test results for increasingly large matrices with dimensions divisible by 16, up to 5 600x5 600.

The CPU version takes about twice as long for the majority of matrices 1 000x1 000 and larger. For matrices smaller than this, the speed increase is less noticeable. For extremely small matrices (smaller than 600x600) the CPU version is faster than the GPU version. The reasoning for this is that using the GPU requires significant constant overhead, which can consume a large percentage of the time [10]. When matrix sizes are increased, the percentage of the total time that the overhead consumes is decreased. This is because the overhead time does not change, but the amount of computation being done significantly increases. Even for matrices near the high end of our selected sizes, the GPU is nearly twice as fast.

Discussion and Conclusion

After testing our algorithm, it became evident which areas needed further research to produce an effective, fully implementable algorithm. The first area to be addressed is the accuracy of the algorithm. Currently, spurious eigenvalues and their resulting vectors are being created due to rounding errors. These are generated during the tridiagonalization process [8], [12], [15]. Two methods may be used in this situation. Either the spurious values can be removed after computation or reorthogonalization can be performed during computation [12], [15]. Removing spurious values after computation, while technically possible, is thought to be less effective than reorthogonalization. Reorthogonalization is the process of ensuring that the generated vectors are an accurate sub-space representation of the original matrix. For this reason, we will aim to implement partial reorthogonalization. Partial reorthogonalization has been proven to be able to preserve semiorthogonality [11], [15]. Partial reorthogonalization would allow us to stop our iteration sequence shorter than a simple Lanczos algorithm. These reasons make a partial reorthogonalization method necessary in a fully implementable LSA algorithm.

Another area of interest in future research would be further increases in the speed of the algorithm. Currently the GPU is only being used for the tridiagonalization computation. However, the *sstev* BLAS routine, which computes the eigenvalues and eigenvectors of a tridiagonal matrix, is still being implemented on the CPU. Transitioning this to the GPU should further increase the performance of the algorithm. The percentage of the total computation time that *sstev* occupies is only around 5% (for a 4 000x4 000 matrix). As matrix sizes grow, the total time that *sstev* requires will also grow, making a GPU version of the routine highly useful. This is especially true if the matrix sizes are approaching 10 000x10 000. Also, improvements upon the CUBLAS *sgemv* routine, which is a matrix-vector multiplication routine, would greatly increase performance as 90% of the GPU computation time is being occupied by this routine. This is entirely feasible as CUBLAS is not stated as being the optimal implementation.

A final area of research would be to attempt to implement the algorithm on a multiple GPU machine. Currently, in order to use multiple GPUs, the algorithm must explicitly state how to use the GPUs. There is no automatic optimization allowing the use of multiple GPUs. For this reason, code needs to be modified for each individual problem. The use of multiple GPUs would be highly beneficial due to the increase of raw computational power. Numerous computers are being released with multiple graphics cards built in or with the capability to add a second graphics card. Adding multi-GPU functionality can be tricky and does not always increase the speed of code because very little effective communication happens between GPUs. This makes sharing computation of problems very difficult, such as when computing a matrix-matrix product. In order to complete the computation, a GPU must have each matrix in its global memory. This adds a lot of increased overhead, as copying memory between GPUs currently requires copying from GPU 1 to the computer then from GPU 2 to the computer. This is very slow and limits the effectiveness of multi-GPU implementation. Further research would be needed to investigate multi-GPU capabilities.

The select few matrices that seem to be performing significantly better than average, specifically matrices with a rank of 2 000, 3 200 and 4 000, are a result of the unique architecture of the GPU. These select few matrices all have one thing in common: The matrix dimensions are all divisible by 16. The reason behind the significant speed increase lies in the architecture of the GPU. When the matrix is not divisible by 16, there are conflicts in shared memory regarding multiple threads accessing the same bank at the same time. This forces one thread into a queue while the other thread is accessing the memory, increasing the total amount of time for all memory accesses. This can be solved by using matrices with dimensions divisible by 16. CUBLAS will as a result provide coalesced memory access patterns, greatly reducing time of the overall function. This was not evident until after obtaining the results, which led us to further test the implementations to verify problem with memory access patterns.

In this research, we developed a parallel latent semantic analysis algorithm for the GPU. The results of our tests are very promising. The speed increase of the GPU based algorithm was five to seven times for matrices with dimensions divisible by 16 and two times for matrices of other sizes. One solution to ensuring that all matrices have dimensions divisible by 16 is to add extra columns and rows of zeros to the matrix. The number of rows and columns to add would be equal to ($\boldsymbol{x}$ modulo 16) where $\boldsymbol{x}$ is equal to the dimension size of the matrix. This number would always be between 1 and 15, requiring minimal computation time when adding this data. We hypothesize that adding these rows and columns of zeros would not add a noticeable increase to computation time and thus would still yield a speed increase of five to seven times. With the GPU being so widespread in modern PCs, this algorithm is not just limited to expensive custom workstations. Most mid-range computers currently come with a graphics card, including laptops. This makes it possible to perform GPU-based LSA on a mobile computer. That being said, a top of the line desktop computer would be expected to see even better results. Currently, the NVIDIA® 280GTX has a theoretical computation level of about 1 teraflop. This is more than twice that of the card we used and costs approximately 115% more. With GPU computational power increasing at a higher rate than CPU computational power, it is very possible to see increased speed results in the near future [10]. We have shown that the GPU can be used to provide a performance increase to our algorithm. This should not only be useful to us, but it will provide evidence to further algorithmic development of the GPU.

Acknowledgements

This research was done at ORNL as part of the U.S. Department of Energy's Student Undergraduate Laboratory Internship (SULI) program. I would like to thank Xiaohui Cui for his support and knowledge of information analyses as well as Engin Sungur for his knowledge of linear algebra. I would also like to thank the Applied Software Engineering Research group as part of the Computational Sciences and Engineering division for their support and assistance throughout the summer. Finally, I would like to thank the Department of Energy and ORNL for giving me this opportunity.

References

[1] N.M. Adams, G. Blunt, D.J. Hand and M.G. Kelly, "Data mining for fun and profit," *Statistical Science*, vol. 15, no. 2, pp. 111–131, 2000.

[2] M.W. Berry, "Large-scale sparse singular value computations," *The International Journal of Supercomputer Applications*, vol. 6, no. 1, pp. 13–49, 1992.

[3] S. Dumais, G. Furnas, T. Landauer, R. Harshman and S. Deerwester, "Using Latent Semantic Analysis to improve access to textual information," in *Proceedings of the Conference on Human Factors in Computing Systems*, 1988, pp. 281–285.

[4] N. Galoppo, N.K. Govindaraju, M. Henson and D. Manocha, "Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware," *ACM/IEEE SC 05 Conference*, vol. A, pp. 3, 2005.

[5] N.E. Gibbs and W.G. Poole, "Tridiagonalization by permutations," *Communications of the ACM*, vol. 17, no. 1, pp. 20–24, 1974.

[6] S. Huang, M.O. Ward and E.A. Rundensteiner, "Exploration of dimensionality reduction for text visualization," Worcester Polytechnic Institute Computer Science Department, Worcester, MA, Technical Report TR-03-14, 2003.

[7] H. Kersken and U. Kuster, "A Parallel Lanczos Algorithm for Eigensystem Calculation," Computing Center of the University of Stuttgart, Stuttgart Germany, 1999.

[8] C. Lanczos, "An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators," *Journal of Research of the National Bureau of Standards*, vol. 45, pp. 255–282, 1950.

[9] S.A. Manavaski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics (Italian Society of Bioinformatics: Annual Meeting)*, vol. 9, pp. S10, 2007.

[10] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn and T.J. Purcell, "A survey of General-Purpose Computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[11] C.C. Paige, B.N. Parlett and H.A. Van der Vorst, "Approximate solutions and eigenvalue bounds from Krylov subspaces," *Numerical Linear Algebra with Applications*, vol. 2, no. 2, pp. 115–134, 1995.

[12] B.N. Parlett and D.S. Scott, "The Lanczos algorithm with selective orthogonalization," *Mathematics of Computation*, vol. 33, no. 145, pp. 217–238, 1979.

[13] P.C.D. Robert, S. Schoepke and H. Bieri. "Hybrid ray tracing — ray tracing using gpu-accelerated image-space methods," presented at International Conference on Computer Graphics Theory, Barcelona, Spain, 2007.

[14] G. Salton and C. Buckley, "Term-Weighting approaches in automatic text retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.

[15] H.D. Simon, "The Lanczos Algorithm with Partial Reorthogonalization," *Mathematics of Computation*, vol. 42, no. 165, pp. 115–142, 1984.