Performance Analysis of CUDA by Implementation of Hypergraph Partitioning for Parallelizing Sparse Matrix-Vector Multiplication Using Quadro K4200

Murni* and T Handhika

Computational Mathematics Study Center, Gunadarma University, 100 Margonda Raya Street, Pondok Cina, Depok, West Java, 16424, Indonesia

*E-mail: murnipskm@staff.gunadarma.ac.id

Abstract. This paper addresses the performance analysis of sparse matrix-vector multiplication through hypergraph partitioning techniques using CUDA GPU-based parallel computing. Quadro K4200 is the GPU that is used in this paper. On the implementation of matrix-vector multiplication, various sizes and types of matrices are attempted. Our results show that on the average scenarios with 2 partitions, 4 partitions, 8 partitions, 16 partitions and 32 partitions in 1024 threads, CUDA performs up to 700 \times better than sequential programming.

1. Introduction

CUDA (Compute Unified Device Architecture) is one of the parallel computing platforms and programming models that have been developed by NVIDIA for general computing on the graphical processing unit (GPU). CUDA was first introduced NVIDIA in 2006. As one of the graphics processor companies, the company is motivated to develop a platform that can be used to maximize GPU usage. This platform is named GPGPU (General Purpose Computation on GPU). However, due to constrained API, then in 2007 NVIDIA issued a parallel programming standard model on a GPU called CUDA. CUDA is a minimal extension of the C / C ++ programming language to direct the process of computing from the CPU to GPU (GPU computing). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPU. Since then, CUDA GPU-based parallel programming has been one of the effective solutions to the constraints faced in MPI and OpenMP implementations, where the resulting speed-up reaches tens to hundreds of times [1].

GPU implements a multi-core architecture consisting of thousands of threads on hundreds of cores. This is certainly different from the CPU that implements multi-core architecture consisting of several cores of multiples of 2. With thousands of threads on hundreds of cores in the GPU, massive parallel programming can be applied to achieve greater performance improvements [4].

This paper will discuss the implementation of sparse matrix-vector multiplication. This implementation based on hypergraph partitioning models in shared memory parallel programming [2, 3, 12, 13, 14]. The benefits obtained by applying hypergraph partitioning using shared memory architecture can minimize communication and cache coherence overhead. The algorithm used in this paper in determining partition of hypergraph partitioning is the Fiducia-Matheyesses-Sanchis algorithm (or known as the FMS algorithm). After the number of partitions is formed, then the matrix problem will turn into a block of matrix. After that, parallel calculations performed on the GPU using CUDA.
2. Hypergraph Partitioning

Hypergraph partitioning is a difficult optimization problem. Then, for solving this problem we can use optimization algorithms. The algorithm used in this paper is KL and FM algorithms developed by Sanchis (or known as FMS algorithms) [5,11]. FMS algorithm is able to partition to K-way partition [15]. The concept of this FMS algorithm is to swap out one by one vertex on each iteration. First of all, define the initial state, i.e. the set of vertices are randomly partitioned into K parts. Furthermore, each vertex will change position from one partition to another partition with the concept of gain, which is a value indicating a reduction cutsize. The positive gain will reduce cutsize while the negative gain will increase cutsize. Vertex with the greatest gain will be moved to the other position. Then, vertex has been moved to other partitions and is not included in the calculation of the gain on the next iteration. In addition, the movement is causing changes in the value of the gain on the other vertices. This process is carried out continuously repeated such that there are no vertex switching positions again. Thus, the optimal solution of the FMS algorithm is the partition with the smallest cutsize.

Hypergraph partitioning will be implemented on the parallelization of sparse matrix-vector multiplication. The basic technique to parallelize sparse matrix-vector multiplication is a way to partition the rows and columns of the matrix into parts. This paper will discuss the process of calculating the multiplication of sub-matrix and sub-vector as follow:

\[ A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \]

In the column-net model, matrix A is represented as a hypergraph \( H_R = (V_R, N_C) \) where each row of the matrix A is represented by the vertices in \( V_R \) and each column of the matrix A is represented by nets in \( N_C \), whereas, in a row-net model, matrix A is represented as a hypergraph \( H_C = (V_C, N_R) \) where each row of the matrix A is represented by the vertices in \( V_C \) and each column of the matrix A is represented by nets in \( N_R \). Thus, there is one vertex \( v_i \) and one net \( n_j \) for each row \( i \) and column \( j \) in column-net or row-net model.

Matrix A subsequently converted into blocks as \( K \times K \) matrix as much by adopting hypergraph partitioning techniques [6, 7, 8, 9]. Block matrix A can be represented as follows.

\[ A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1K} \\ A_{21} & A_{22} & \cdots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \cdots & A_{KK} \end{bmatrix}, \]

where \( K \) is the number of processors with the size of each matrix A is \( m_i \times n_j \), such that \( \sum_{i=1}^{K} m_i = m \) dan \( \sum_{j=1}^{K} n_j = n \).

Furthermore, vector \( \vec{x} \) also divided into several sub-vectors corresponding as follows:

\[ \vec{x} = \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_K \end{bmatrix} \]

where \( \vec{x}_i \) consists of \( n_j \) elements (\( \sum_{i=1}^{K} n_i = n \)).

In hypergraph partitioning techniques, blocks of matrix A is not located on the main diagonal are arranged such that the majority of the elements of the matrix block a lot of zeroes. It means that communication between processors is reduced. Furthermore, submatrix A and subvector \( \vec{x} \) are multiplied in parallel using CUDA on GPU.

Problems matrix A and the vector \( x \) above have been simplified using hypergraph partitioning into sub-matrices A and sub-vector \( \vec{x} \) with the size \( K \times K \) and \( K \times 1 \). Thus, the multiplication of sub-matrix A and sub-vector \( x \) will produce \( K \times 1 \) sub-vector \( \vec{y} \), as given in the following equation:
The next procedure on the parallelization of sparse matrix-vector multiplication using hypergraph partitioning on the GPU is determining the number of blocks. Determination of the amount of this block in accordance with the size of the multiplication result sub-matrix $A$ with sub-vector $x$, that is $K \times 1$. This is what distinguishes the procedure on the GPU parallelization of matrix multiplication. The number of block-based parallelization procedures of sparse matrix multiplication on GPU determined from the maximum limit thread owned machine GPU, while the number of block-based procedures parallelization of sparse matrix multiplication using hypergraph partitioning on the GPU is determined from the size of the sub-matrix multiplication results. After that, the procedure followed was to determine the total of the block size for each block. The number of threads in one block should not exceed the number of threads that exist on the machine GPU. If the thread in one block exceeds the number of threads that exist on the machine GPU then the process will occur sequentially in one block. To overcome this, the number of partitions to be enlarged so that the number of blocks obtained will be more and more. This is done repeatedly to obtain the block number where the number of threads in each block meets the existing number of threads on the GPU. Once this is completed then the process parallelization of sparse matrix-vector multiplication using hypergraph partitioning can be processed using CUDA on GPU [10, 14].

3. Implementation

In the GPU there is a grid, block and thread hierarchy [16]. Therefore, before proceeding the sparse matrix-vector multiplication procedure, it is necessary to explain the specifications CPU and GPU to be used. The parallelization program of matrix-vector multiplication will be run on the CPU and GPU with the following specifications:

**CPU:**
- Model Identifier: ASUS MAXIMUS VII HERO
- Processor Name: Intel Core i7
- Processor Speed: 3.6 GHz
- Number of Processor: 1
- Total Number of Cores: 4
- L2 Cache (per Core): 256 KB
- L3 Cache: 6 MB
- System Bus: Intel Direct Media Interface 5GT/s
- Software: OS X 10.9.1 (13B42)

**GPU:**
- Mesin: Quadro K4200
- Architecture: Kepler
- CUDA Cores: 1344

For Quadro K4200 is known that machines used have the Kepler architecture, a block may have at most 1024 threads. Given a various matrix $A$ and vector $x$. In this paper, five scenarios have been done, such as Hypergraph Partitioning with 2 partitions, 4 partitions, 8 partitions, 16 partitions and 32 partitions. All these scenarios use 1024 threads. The results can be seen in the following figure.
Figure 1: Hypergraph Partitioning with 2 Partitions

Figure 2: Hypergraph Partitioning with 4 Partitions

Figure 3: Hypergraph Partitioning with 8 Partitions
Figure 4: Hypergraph Partitioning with 16 Partitions

Figure 5: Hypergraph Partitioning with 32 Partitions

Based on the results in Figure 1, Figure 2, Figure 3, Figure 4 and Figure 5 we find that the fastest speed up for Sparse Matrix-Vector Multiplication in this study is obtained in Figure 3, which is Hypergraph Partitioning with 8 partitions. The speed-up movement increases with the increasing size of the matrix. Overall we achieved CUDA performs up to \( \times 700 \) better than sequential programming.

4. Conclusion and future works

Based on the discussion of this paper, it can be concluded that the Hypergraph Partitioning in shared-memory parallel programming in sparse matrix-vector multiplication using CUDA with GPU can perform up to \( 700 \times \) better than sequential programming. This result is obtained by using five scenarios on various matrices. The speed-up movement increases with the increasing size of the matrix.

As future work, the use of different types of GPU devices is possible, more high specification to process the more large matrix that used. Furthermore, the parallel reduction can be performed on the parallelization of sparse matrix-vector multiplication to obtain better performance.

References
[1] A. Bustamam, G. Ardaswari, H. Tasman and D. Lestari. 2014. Performance Evaluation of Fast Smith-Waterman Algorithm for Sequence Database Searches using CUDA GPU-based Parallel Computing, Journal of Next Generation Information Technology, 5, pp 38-46
[2] U.V. Catalyurek and C. Aykanat. 1996. Decomposing Irregularly Sparse Matrices for Parallel Matrix-Vector Multiplication, Parallel Algorithms for Irregularly Structured Problems, Irregular '96. In Lecture Notes in Computer Science, 1117, pp 75-86
[3] U.V. Catalyurek and C. Aykanat. 1999. Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication, IEEE Transactions on Parallel and Distributed Systems, 10, pp 673-693
[4] S. J. Eggers. 1991. Simplicity Versus Accuracy in a Model of Cache Coherency Overhead, IEEE Transactions on Computer, 40, pp 893-906
[5] C.M. Fiduccia and R.M. Mattheyses. 1982. A Linear-Time Heuristic for Improving Network Partitions. Proc. 19th ACM/IEEE Design Automation Conf., pp 175-181
[6] B. Hendrickson. 1998. Graph Partitioning and Parallel Solvers: Has the Emperor No Clothes?, Solving Irregularly Structured Problems in Parallel, Irregular '98, Number 1457 In Lecture Notes in Computer Science, pp 218-225
[7] B. Hendrickson and T.G. Kolda. 1998. Partitioning Sparse Rectangular Matrices for Parallel Computations of Ax and A’x, Applied Parallel Computing in Large Scale Scientific and Industrial Problems, PARA ’98, Number 1541 In Lecture Notes in Computer Science, 1541, pp 239-247
[8] B. Hendrickson and T.G. Kolda. 2000. Graph Partitioning Models for Parallel Computing, Parallel Computing, 26, pp 1519-1534
[9] B. Hendrickson and T.G. Kolda. 2000. Partitioning Rectangular and Structurally Unsymmetric Sparse Matrices for Parallel Processing, SIAM J., 21, pp 2048-2072
[10] R. Hochberg. 2012. Matrix Multiplication with CUDA – A basic introduction to the CUDA programming model, Shodor.
[11] B.W. Kernighan and S. Lin. 1969. An Efficient Heuristic Procedure for Partitioning Graphs, The Bell System Technical Journal, pp 291-307
[12] Murni, T. Handhika, I. Sari and D. Indarti. 2016. Implementasi Graph Partitioning pada Paralelisasi Perkalian Matriks-Vektor, in Proceedings of National Seminar in Mathematics and Statistics (SEMASTAT) Universitas Negeri Padang, pp 194-199
[13] Murni and T. Handhika. 2016. Implementasi Hypergraph Partitioning pada Paralelisasi Perkalian Matriks-Vektor, in Journal of Computer & Information System, pp 1-6
[14] Murni, A. Bustamam , Ernastuti, T. Handhika and Kerami D. 2016. Hypergraph partitioning implementation for parallelizing matrix-vector multiplication using CUDA GPU-based parallel computing, in Proceedings of the international symposium on current progress in mathematics and sciences, pp 1-6
[15] L.A. Sanchis. 1989. Multiple-Way Network Partitioning. IEEE, 38, pp 62-81
[16] J. Sanders and E. Kandrot. 2008. CUDA by Example: An Introduction to General Purpose GPU Programming, New Jersey, Addison-Wesley