Hypergraph partitioning implementation for parallelizing matrix-vector multiplication using CUDA GPU-based parallel computing

Cite as: AIP Conference Proceedings 1862, 030153 (2017); https://doi.org/10.1063/1.4991257
Published Online: 10 July 2017

Murni, A. Bustamam, Ernstutti, T. Handhika, and D. Kerami

ARTICLES YOU MAY BE INTERESTED IN

Data preprocessing for determining outer/inner parallelization in the nested loop problem using OpenMP
AIP Conference Proceedings 1862, 030138 (2017); https://doi.org/10.1063/1.4991242

Parallelization strategies for continuum-generalized method of moments on the multi-thread systems
AIP Conference Proceedings 1862, 030146 (2017); https://doi.org/10.1063/1.4991250

Clustering self-organizing maps (SOM) method for human papillomavirus (HPV) DNA as the main cause of cervical cancer disease
AIP Conference Proceedings 1862, 030155 (2017); https://doi.org/10.1063/1.4991259
Hypergraph Partitioning Implementation for Parallelizing Matrix-Vector Multiplication Using CUDA GPU-Based Parallel Computing

Murni1 a), A. Bustamam2, Ernstutti1, T. Handhika1, and D. Kerami2

1Computational Mathematics Study Center, Gunadarma University, Depok, Indonesia
2Department of Mathematics, Faculty of Mathematics and Natural Sciences (FMIPA), Universitas Indonesia, Depok 16424, Indonesia

a)Corresponding author: murnipskm@staff.gunadarma.ac.id

Abstract. Calculation of the matrix-vector multiplication in the real-world problems often involves large matrix with arbitrary size. Therefore, parallelization is needed to speed up the calculation process that usually takes a long time. Graph partitioning techniques that have been discussed in the previous studies cannot be used to complete the parallelized calculation of matrix-vector multiplication with arbitrary size. This is due to the assumption of graph partitioning techniques that can only solve the square and symmetric matrix. Hypergraph partitioning techniques will overcome the shortcomings of the graph partitioning technique. This paper addresses the efficient parallelization of matrix-vector multiplication through hypergraph partitioning techniques using CUDA GPU-based parallel computing. CUDA (compute unified device architecture) is a parallel computing platform and programming model that was created by NVIDIA and implemented by the GPU (graphics processing unit).

Keywords: graph partitioning, hypergraph partitioning, parallelization, matrix-vector, CUDA

INTRODUCTION

In a real world, calculation of matrix-vector multiplication is needed in solving various problems, for example in determining the solution of linear equations system. The large size of matrix constraints led to a long time of the calculation process. This results in an accuracy of calculations that are not real-time. Several previous studies have tried to do the parallelization of the calculation of the matrix-vector multiplication either by adopting the concept of the graph, which is a graph partitioning [1]. However, there is an assumption that must be met in implementing the techniques of graph partitioning. The matrix must be perfectly square and symmetrical. In fact, the problems of the real world, the size of the matrix is not limited to a square and symmetrical only.

In this paper, the authors use hypergraph partitioning techniques to overcome the shortcomings of graph partitioning techniques with which this technique can be used for any size of the matrix. Unlike the graph partitioning techniques, every edge in hypergraph partitioning technique can connect more than two vertexes or commonly refer to as net (hyperedge).

Hypergraph partitioning techniques that have been discussed in previous studies have been implemented with a multi-processor programming model on distributed memory architectures [2–3]. So in this paper will try for shared-memory architecture. In the shared memory architecture, a computer designed with a hierarchical memory system that is small, inexpensive and rapid known as cache memory. The memory size of the cache memory is much smaller than the main memory impedes the distribution of data. So that the computing process on each core becomes longer. Moreover, another problem that arises is resulting cache coherence mechanism, the mechanism used by the system to ensure that the processor does not access the wrong data. With the cache coherence, then there is the cost
incurred, i.e. cache coherence overhead. Based on the description of the cache memory, it can be concluded that the management of cache memory is very important to consider in parallel programming.

Thus, in this paper will discuss the implementation of hypergraph partitioning models in shared memory parallel programming for parallelizing sparse matrix-vector multiplication. 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 the partition of hypergraph partitioning is Fiducia-Matheyesses-Sanchis algorithm (or known as FMS algorithm). After the number of partition is formed, then the matrix problem will turn into a block of the matrix. After that, parallel calculations performed on the GPU using CUDA.

**HYPERGRAPH PARTITIONING**

Suppose that undirected hypergraph denoted as $\mathcal{H} = (V, N)$ with $V$ is not an empty set of vertices where $n = |V|$ is the number of vertices. At the same time, $N$ is a set of nets that connecting the vertices. Every net, $n_j \in N$ is a subset of vertices ($n_j \subseteq V$). The vertices in a net $n_j$ are called its pins and denoted as $pins[n_j]$. The size of a net is equal to the number of its pins and denoted as $s_j = |pits[n_j]|$ [2-3]. The set of nets connected to a vertex $v_l$ is denoted as $nets[v_l]$. Furthermore, the degree of a vertex is equal to the number of nets it is connected to, i.e., $d_l = |nets[v_j]|$. The graph is a special case of a hypergraph such that each net has exactly two pins. As in the graph, the weights of vertex and the cost of the net also defined in the hypergraph. Let $w_i$ and $c_j$ denote the weight of vertex $v_1 \in V$ and the cost of the net $n_j \in N$ [2-3].

Let $\Pi = \{P_1, P_2, \ldots, P_K\}$ is a $K$-way partition of $\mathcal{H}$. A net that has at least one pin (vertex) in a part is said to connect that part. Connectivity set $\Lambda_j$ of a net $n_j$ is defined as the set of parts connected by $n_j$. The number of partitions that are connected by $n_j$ is denoted by $\lambda_j = |\Lambda_j|$. A net $n_j$ is said to be cut if it connects more than one part ($\lambda_j > 1$) and uncut, otherwise ($\lambda_j = 1$). Cut and uncut nets are also called the external and internal net. The set of external nets of a partition $\Pi$ is denoted as $E$. Furthermore, given the two definitions of the cut size to represent the total cost to the partition $\Pi$ as [2-3]:

$$\chi(\Pi) = \sum_{n_j \in E} c_j$$

and

$$\chi(\Pi) = \sum_{n_j \in E} c_j (\lambda_j - 1)$$

In (1), each cut net $n_j$ contributes $c_j$ to the cut size. While, in (2), each cut net $n_j$ contributes $c_j(\lambda_j - 1)$ to the cut size. Thus, the problem of hypergraph partitioning can be defined as a technique for partitioning a hypergraph into two or more parts so that cut size minimum is obtained with the balance criterion as follows [2-3]:

$$W_k \leq W_{avg}(1 + \epsilon), \text{ for } k = 1, 2, \ldots, K$$

where

$$W_k = \sum_{v_i \in P_k} w_i \text{ and } W_{avg} = \frac{\sum_{v_i} w_i}{K}$$

and $\epsilon$ represents the predetermined maximum imbalance ratio allowed.

Therefore, in this paper will discuss the hypergraph partitioning on the model of shared-memory parallel programming, then there is a cache coherence overhead. The three components that affect the cache coherence overhead are cache coherence block size (thread), cache size, and type of bus operation [4]. Thus, the hypergraph partitioning models in shared-memory parallel programming can be used to reduce the communication between the processor and cache coherence overhead.

Once the model hypergraph partitioning is formed, the next stage is to complete the solution of the model. 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 [7] (or known as FMS algorithms) [5-6]. FMS algorithm is able to partition to $K$-way partition [7]. The concept of this FMS algorithm is 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 cut size. The positive gain will reduce cut size while the negative gain will increase cut size. Vertex with the greatest gain will be moved to the other position. Then, a vertex
Parallelization Procedure of Sparse Matrix-Vector Multiplication Using Hypergraph Partitioning

Hypergraph partitioning discussed in the previous sections will be implemented on 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 sparse matrix-vector multiplication in parallel with adopting hypergraph partitioning techniques.

In hypergraph partitioning, there are two models for decomposition of a sparse matrix. These models are column-net and row-net models. Suppose given \( F \times V \) matrix and \( V \times L \) vector as follows:

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

In the column-net model, the matrix \( A \) is represented as hypergraph \( \mathcal{H}_R = (V_R, N_R) \), 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 \( N_R \). Whereas, in the row-net model, the matrix \( A \) is represented as hypergraph \( \mathcal{H}_C = (V_C, N_C) \) 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 \( N_C \). Thus, there is one vertex \( x_i \) 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 [1]. Here are the steps used to obtain the matrix blocks.

1. Rows of the matrix \( A \) divided into \( K \) partition by adopting techniques hypergraph partitioning algorithms using FMS.
2. Columns of the matrix \( A \) divided into \( K \) partition by adopting techniques hypergraph partitioning algorithms using FMS.
3. Arrange the rows of the matrix partition results in step 1, respectively, starting from the rows of the matrix into the first partition until the rows of the matrix into the partition \( K \).
4. Arrange the columns of the partition on the results of step 1, respectively, starting from the columns of the matrix into the first partition until the columns of the entrance into the partition \( K \).
5. Finally, the matrix \( A \) is formed into blocks as \( K \times K \) matrix.

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

\[
\sum_{i=1}^{K} m_i = m \quad \text{and} \quad \sum_{j=1}^{K} n_j = n.
\]

Furthermore, the 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 \( n_i \) elements \( \left( \sum_{i=1}^{K} n_i = n \right) \).
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 $\bar{x}$ are multiplied in parallel using CUDA on GPU.

RESULTS AND DISCUSSION

There is a procedure to perform parallelization of sparse matrix-vector multiplication using hypergraph partitioning on the GPU. The following will explain the parallelization procedure of sparse matrix multiplication using hypergraph partitioning on GPU.

Suppose given $m \times n$ matrix $A$ and $n \times 1$ vector $\bar{x}$. Furthermore, suppose the result of multiplying matrix $A$ by the vector $\bar{x}$ is a vector $\bar{y}$ with the size $m \times 1$.

In the GPU there is a grid, block and thread hierarchy [8]. Therefore, before proceeding 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 (Table 1).

For GeForce GT-650M is known that machines used have the Fermi architecture, a block may have at most 1024 threads. Thus if we are multiplying a $m \times n$ matrix $A$ by $n \times 1$ vector $\bar{x}$, we would be launching $m$ threads total to compute the entries of the $m \times 1$ product matrix. Furthermore, the threads on CUDA are grouped into several blocks in which a block can have a maximum of 1024 thread. Threads within the same block have access to very fast "shared memory" which is shared the way threads of a process share memory on an ordinary computer, while threads in different blocks must communicate via off-chip "global memory,” which is much slower [9].

Problems matrix $A$ and the vector $\bar{x}$ above have been simplified using hypergraph partitioning into sub-matrices $A$ and sub-vector $\bar{x}$ with the size $K \times K$ and $K \times 1$. Thus, the multiplication of sub-matrix $A$ and sub-vector $\bar{x}$ will produce $K \times 1$ sub-vector $\bar{y}$, as given by the following equation:

$$\bar{y} = \begin{bmatrix}
\bar{y}_{11} \\
\bar{y}_{21} \\
\vdots \\
\bar{y}_{K1}
\end{bmatrix}.$$

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 subvector $\bar{x}$, 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.

### TABLE 1. Specification of CPU and GPU

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Identifier : MacBookPro 10,1</td>
<td>Engine : GT 650M</td>
</tr>
<tr>
<td>Processor Name : Intel Core i7</td>
<td>Architect : Fermi</td>
</tr>
<tr>
<td>Processor Speed : 2.3 GHz</td>
<td>CUDA Cores : 384</td>
</tr>
<tr>
<td>Number of Processor : 1</td>
<td></td>
</tr>
<tr>
<td>Total Number of Cores : 4</td>
<td></td>
</tr>
<tr>
<td>L2 Cache (per Core) : 256 KB</td>
<td></td>
</tr>
<tr>
<td>L3 Cache : 6 MB</td>
<td></td>
</tr>
<tr>
<td>System Bus : Intel Direct Media Interface 5GT/s</td>
<td></td>
</tr>
<tr>
<td>Software : OS X 10.9.5 (13F1911)</td>
<td></td>
</tr>
</tbody>
</table>

030153-4
After that, the procedure followed was to determine the total of the blocksize for each block. The total of blocksize used in each block is a maximum of the number of rows of sub-matrix $A$ multiplied by the number of the sub-column vector $\bar{x}$. The blocksize number on each block can be formulated as follows.

$$\text{blocksize} = \max\{m_i e_{A_K \times x_K} \times 1\},$$

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.

For example, given $100 \times 100$ matrix $A$ and $100 \times 1$ vector $\bar{x}$. Furthermore, the matrix $A$ is partitioned into two submatrix with each size is $50 \times 50$. The results can be seen in Table 2.

According to the Table 2 above shows that the fastest time obtained when the total of blocksize equal to the maximum of the number of rows of sub-matrix $A$ multiplied by the number of the sub-column vector $x$.

**CONCLUSIONS AND FUTURE RESEARCH**

Based on the discussion of this paper, it can be concluded that the model building hypergraph partitioning in a shared-memory parallel programming in sparse matrix-vector multiplication can be achieved. If the model of distributed-memory parallel programming, hypergraph partitioning techniques can only minimize communication between processors but not on the model of shared-memory parallel programming. The advantages derived from hypergraph partitioning techniques in a shared-memory parallel programming in sparse matrix-vector multiplication with the GPU in addition to minimizing the communication between the processor also can minimize the cache coherence overhead.

Hypergraph partitioning in a shared-memory parallel programming in sparse matrix-vector multiplication using CUDA with a GPU not taking into account the efficiency of programming. Thus, this programming may be developed taking into account the efficiency of the programming.

**REFERENCES**