ADDIS ABABA UNIVERSITY

School Of Electrical And Computer Engineering

Improve HMC Based Graph Processing By Adding Compress/Decompress Unit

by

Betelhem Mengesha

Advisor: Dr. Fitsum Assamnew

A thesis submitted to Addis Ababa Institute of Technology, School of Graduate Studies, Addis Ababa University in partial fulfillment of the requirements for the Degree of Master of Science in Computer Engineering

May 2022

Declaration

I, the undersigned, declare that this MSc thesis is my original work, has not been submitted in part or in whole, at any other university and all sources, and materials used for the thesis work, and the organizations and individuals who contributed a lot during the development of this study have been acknowledged. This research was conducted at Addis Ababa University under the supervision of Dr. Fitsum Assamnew

Submitted by:

. . . .

Betelhem Mengesha

ID No. GSR/0134/10

Acknowledgements

The success of this study is from a wide range of contributions. First and foremost, I would like to give all honor and glory to the almighty God.

I would like to express my deep and sincere gratitude to my advisor, Dr.-Fitsum Asaminew for his guidance, encouragement, inspiration and positive attitude during my study. Without his continuous help, I would not have been able to complete this task. In addition, his comments to the entire development of what the research would look like and the contents to be covered in the study were very fruitful for making this thesis to come to be true.

I also like to thank my parents for their love and continuous encouragement to do the best possible.

Abstract

Graphs play an important role in various practical application areas from social science to machine learning. However, due to the irregular data access pattern of graph computation, there is a major challenge in graph processing.
The emergence of the technology called Hybrid memory cube(HMC) has helped graph processing accelerators to overcome this issue. This hardware provides efficient bandwidth to the graph computation, however, the communication traffic between memory cubes limits the performance. To overcome this issue we proposed a new approach for HMCs based accelerators by adding a packet compression/ decompression unit. We used Message Fussion and Tesseract as our baseline system. In our approach, the data sent between the memory cubes will be compressed before being sent into the network. From the experimental result, the proposed approach showed 1.7x performance improvement on average over the baseline systems. In addition, the energy consumption by the transmission of the network is reduced by 47.28% over the baseline system and the compressor/decompressor unit takes 25% of the total area.

Contents

Declaration i

Acknowledgements ii

Contents iv

List of Figures vi

List of Tables vii

Abbreviations viii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objective . . . 2
  - 1.2.1 General Objective . . . 2
  - 1.2.2 Specific Objective . . . 3
  - 1.2.3 Research Methodology . . . 3
    - 1.2.3.1 Research Setting . . . 3
- 1.3 Scope . . . 3
- 1.4 Contribution . . . 4
- 1.5 Thesis Organization . . . 4

2 Background 5
- 2.1 Graph representation . . . 5
- 2.2 Graph Representation on Memory . . . 5
- 2.3 HMC Architecture . . . 7
- 2.4 Graph Data Partitioning . . . 7
- 2.5 Network Topology . . . 8
- 2.6 Graph Algorithm . . . 9
- 2.7 Data Compression Algorithm . . . 10

3 Literature Review 15
- 3.1 Related work on Graph processing Hardware Architecture . . . 16

3.1.1 Conventional System 16
3.1.2 Graph processing on GPU 17
3.1.3 In memory processing 18
3.1.4 External memory support based processing 19
3.2 Related work on Graph compression techniques 20

4 Proposed Methodology **23**
4.1 The baseline system 23
4.2 Our proposed architecture 25
4.3 Hardware simulators 26

5 Experiment **28**
5.1 Experimental setup 28
5.2 Workloads 29
5.3 Evaluations Result 29
5.3.1 Computational speed evaluation 29
5.3.2 Energy and Area Evaluation 33
5.4 Future work 34

6 Conclusion **35**

A Appendix **36**

Bibliography **38**

List of Figures

1 A typical graph and its representation. 6
2 Graph representation on memory 6
3 A typical graph and its representation. 8
4 The two common network topology for connecting HMCs. 9
5 Pseudocode of BFS Algorithm 10
6 Code Tree 11
7 Huffman Algorithm flow chart 13

8 Evolution of hardware architecture [12]. 17
9 Replica representation of vertices 19
10 Graph partitioning methods 19
11 Graph Representation formats 20

12 storage of graph data 24
13 The baseline system architecture [10] 24
14 router architecture of proposed system 25
15 four inputs with five output crossbar switches[20] 25
16 baseline system with helper(compressor) 26

17 Performance improvement over Tesseract[9] and MessageFussion[10] for pagerank algorithm 30
18 Performance improvement over Tesseract[9] and MessageFussion[10] for SPMV algorithm 30
19 Performance improvement over Tesseract[9] and MessageFussion[10] for SSSP algorithm 31
20 Performance improvement over Tesseract[9] and MessageFussion[10] for BFS algorithm 31
21 total number of messages in network for PageRank algorithm 32
22 total number of messages in network for SPMV algorithm 32
23 total number of messages in network for SSSP algorithm 33
24 total number of messages in network for BFS algorithm 33

List of Tables

2 The hardware and software specification of the machine used in this thesis . 28
3 Data sets used in our experiment [21] 29

Abbreviations

HMC	**H**ybrid **M**emory **C**ube
SSSP	**S**ingle **S**ource **S**hort **P**ath Algorithm
BFS	**B**readth **F**irst **S**earch Algorithm
SPMV	**SP**arse **M**atrix **V**ector Multiplication Algorithm

Chapter 1

Introduction

Graphs play an important role in various practical application areas. Most real world data can be represented as graphs for better readability and understanding. Hence graphic data processing is an important subject attracting the attention of researchers. This chapter discusses about the general background information, motivation, problem statement and contributions of this research work.

1.1 Motivation

Most real world data can naturally be represented as graphs, and therefore graphs play important role in various application areas. Some application areas of graph processing are: cyber security [1], social media [2], natural language processing [3, 4, 5], and machine learning [6, 7, 8]. In general, these applications are have high bandwidth requirement, therefore they require efficient way of implementation.

Conventional computer hardware architecture has limitations in graph processing. In the conventional computer hardware structure, the data which will be executed is stored near to processor (cache) for further processing. This approach fails for graph processing,because it puts high pressure on memory. That is, there is high memory communication with low computation in graph processing. In addition there is random access of memory in graph computation. This condition leads to low cache hit rate. This means the cache

can't predict the next memory access. Therefore the use of cache is meaningless in this kind of application.
Another approach emerged was the so called data-centric approach. This approach is also called in memory processing approach. Early works on processing in memory architectures didn't get much of attentions due to lack of appropriate technology and unpracticed data-intensive applications[9]. However; it started to get much attention in recent researchers [9, 10, 11] due to emergence of new 3D memory technologies and wide uses of data intensive applications[9]. In this kind of approach the processor will be near to where data resides. The main advantages of this approach are:- (1) distance between processing unit and memory will be reduced. (2) Scalability of the system increases. There are two common 3D memories technologies; namely, High Bandwidth Memory (HBM) and Hybrid Memory Cube (HMC).
Junwhan et al. (2015) tried to implement graph computation in HMC for the first time and it showed high improvement in computation time over conventional architecture. However, this solution provides limited energy benefit due to high intercommunication between cubes. In addition they tried to show the performance of graph computation in HMC is much lower than ideal HMC memory technology can provide, that is due to high inter cube communication. Therefore; they implied the need of work to fill this gap. Different papers tried to reduce interconnection issue by reducing data communication [11, 10]. This reduction of data in communication showed a huge improvement on computational time. But still it doesn't meet the full potential of what HMC can provide.

1.2 Objective

1.2.1 General Objective

The general objective of this thesis is to improve computational speed and reduce energy consumption of HMC based graph processing accelerators by providing additional component called compress/decompress unit. This unit reduce the inter communication between the cubes by providing compression of data before it sent into network as packet.

1.2.2 Specific Objective

- Understand Huffman and cantor paring data compressing algorithm
- Design packing unit in to the baseline system. This packing unit packs multiple packets in to one.
- Add compressor/decompress unit to baseline architecture and compare the computational time and energy improvement over the baseline system

1.2.3 Research Methodology

A quantitative approach was followed. This research uses correlational study to compare proposed approach with existing graph processing accelerators. Computation time used as comparing parameter of the two systems.
Correlational study was selected to show computation time improvement of proposed system from current system.

1.2.3.1 Research Setting

The study was conducted on multicore machine. We used CasHMC and Booksim to simulate our architecture and our baseline system.

1.3 Scope

The research considers the improvement of HMC-based graph processing accelerators by adding additional component hardware called compress/decompress units. This unit uses to compress data. After compression of the data, the multiple packets can be packed as one before sending it to the network. The research is only conducted for PageRank, Breadth-First Search (BFS), Single source shortest path algorithm (SSSP), and Sparse Matrix-Vector Multiplication (SpMV) algorithms. We used the Huffman encoding algorithm to compress our data. We didn't consider the improvement of the compression algorithm in this thesis and we left it as future work. We compared our results based on computational

speed and energy consumption as the metrics.
The proposed architecture is simulated by custom simulation software built by Booksim and CasHMC. The energy consumption of the system is tested by Orion.

1.4 Contribution

The thesis has the following contributions:

- We add compressor/decompress unit and packing unit to HMC based architecture. This reduces the number of packets send between HMC cubes
- We evaluated our system on four algorithms with four different dataset and we got up to 1.7x performance improvement over our baseline system.In addition the energy consumption by transmission of the network is reduced by 47.28%

1.5 Thesis Organization

The thesis is organized in to six chapters including this introduction as chapter 1. Chapter 2 describes the theoretical background about HMC based accelerators and different graph processing algorithms. Chapter 3 contains the literature review on different graph processing accelerators. In Chapter 4 the proposed approach is presented. Chapter 5 explains the experiment result of the proposed architecture and our baseline system while Chapter 6 is conclusion and recommendations of the work.

Chapter 2

Background

This section discusses about Graph representation, HMC architecture, our baseline architecture, common graph algorithms and the category of our proposed architecture.

2.1 Graph representation

A graph is typically represented as G = (V,E), where V represents the vertex set and E represents the edge set. An Edge of graph can be represented as e = (vi,vj), where vi stands for source vertex and vj stands for destination vertex. A vertex degree of v represents the total number of edges with the source vertex v. For example, the degree of vertex0 in figure 1 is three. There are two type of graph based on edge values. These are weighted and unweighted graph. The weighted graph is a graph that, any of its edge has a cost called weight. On the other hand unweighted graph has edges, which don't have cost or weight. The graph shown in figure 1 is weighted graph and figure 2 is unweighted graph.

2.2 Graph Representation on Memory

There are two common types of graph representations on memory. These are adjacency matrix and adjacency list [13]. In adjacency list representation graphs are stored as a list

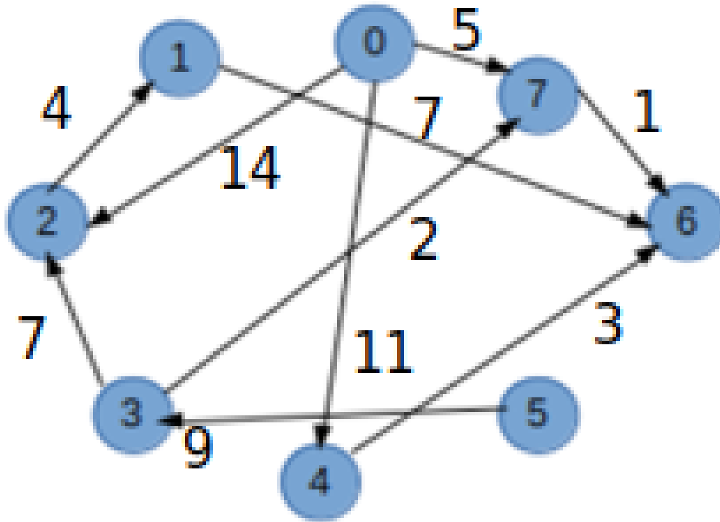
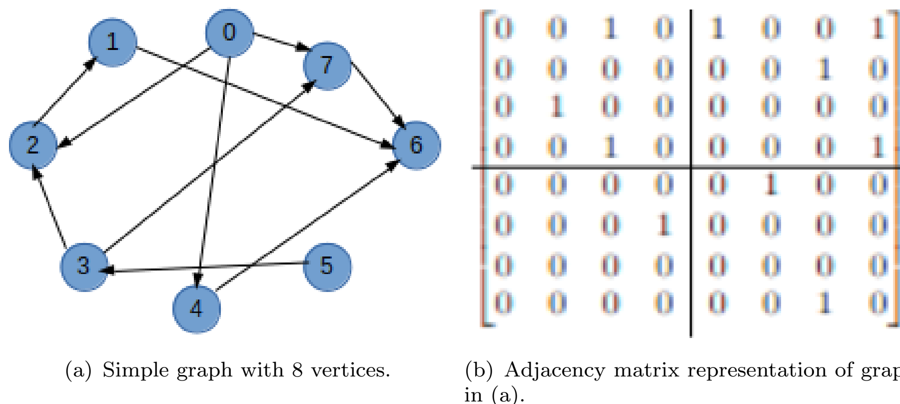


FIGURE 1: A typical graph and its representation.



(a) Simple graph with 8 vertices.

(b) Adjacency matrix representation of graph in (a).

FIGURE 2: Graph representation on memory

of order pairs and values. That is source vertex, destination vertex and graph values. For example data of an edge 1-6 in figure 1 can be represented as (1,6,1). The other way of graph representation is adjacency matrix. In this representation graph data is stored as matrix format as shown in figure 2(b).

We used adjacency list for our representation. This is because matrix representation wastes memory for sparse graphs.

2.3 HMC Architecture

HMC is a 3D memory technology. It is a package of four or eight DRAM die and logic die, which is attached together by silicon via technology (TSV). Within HMC each memory organized as an independent component called vault. A vault is a simple combination of one logic layer and 4 or 8 DRAMs. Logic layer contains memory controller and computing unit. The memory controller is used to control memory read and write operations. And the other component called computing unit performs some tasks on data. For example in graph computation, a graph data will be stored in dram memories and the graph computation will be performed by compute unit, which is located at logic layer. In this research a HMC with a total of 32 vaults are used. The overall architecture of HMC is shown in figure 3.
The main advantage of HMC hardware is they provide higher parallelism of data, provide near memory computing and provides a high bandwidth up to 320GB per second. In addition, due to data transfer can be reduced, the power consumption is reduced. However it has a high inter cube communication issue for data intensive applications such as graph processing. Therefore an efficient implementation of this system is needed.

2.4 Graph Data Partitioning

The graph data from the user is first handled by the CPU. It is The CPU's duty to partition data and distribute them into HMCs. Efficient way of partitioning mechanism is an open research area due to irregular structure of graph data. The main challenges in partitioning graph data are load imbalance issue because of irregular structure of graph, and a high dependence between a graph data [23]. The common graph partitioning methods used in current graph accelerators are source oriented partitioning, destination oriented partitioning and grid partitioning method. The source oriented partitioning method works as follows. First the vertices of the graph are divided in to K parts based on their IDs. The value of K is determined by number of vertices and the memory capacity of graph processing accelerator. And then, each vertices stores its own out-edges. Out-edges of

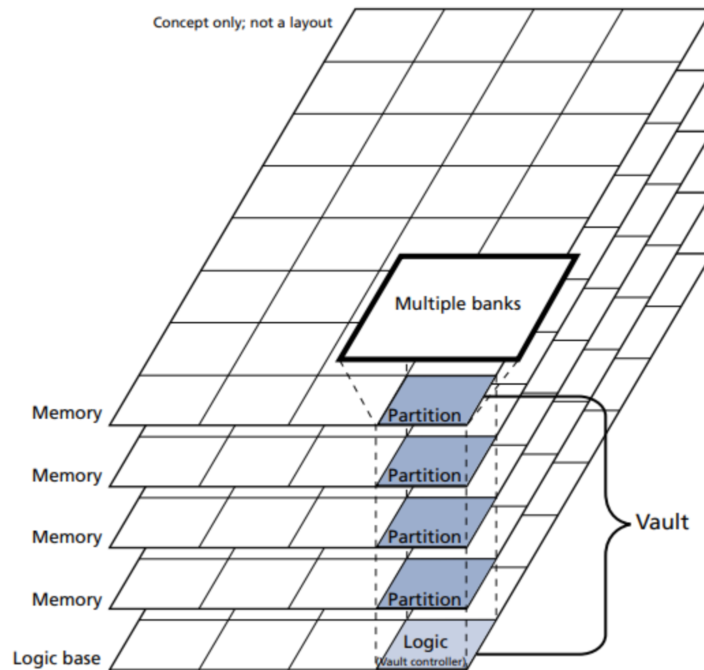
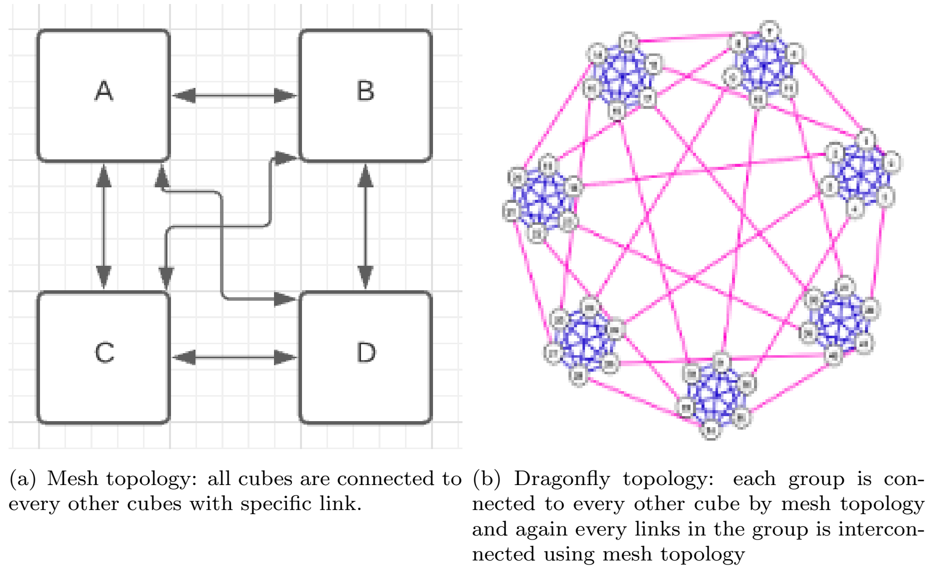


FIGURE 3: A typical graph and its representation.

vertex A is any edge, which its source vertex is vertex A. The other partitioning method called destination oriented partitioning is similar with source oriented partitioning method but the difference is its partitioning scheme is based on destination vertex rather than the source vertex. The other partitioning method called grid partitioning method is a two dimensional partitioning mechanism. That is the partitioning is based on both source and destination vertices.

2.5 Network Topology

Network topology determines the geometric representation of how nodes and routers are connected to each other. In our case the cubes are Hybrid Memory Cubes (HMC). The most common topology used to connect multiple HMCs are mesh topology and dragonfly topology. A mesh architecture is one of the commonly used topologies in HMCs. It can be represented as a k-n cubes where k is the number of routers and n is the number of dimensions. In Mesh topology each cubes connected to all other cubes by specific point to



(a) Mesh topology: all cubes are connected to every other cubes with specific link.

(b) Dragonfly topology: each group is connected to every other cube by mesh topology and again every links in the group is interconnected using mesh topology

FIGURE 4: The two common network topology for connecting HMCs.

point link. It reduces the data traffic issue, due to the dedicated link between any of two terminals [10]. In dragonfly topology the cubes are split into multiple groups [14]. Then each group interconnected to other group using any other topology. An example of these typologies is shown in figure 4.

We used mesh topology to interconnect memory in our system.

2.6 Graph Algorithm

In this thesis, four common graph algorithms are used. These are PageRank, Single Source Shortest Path (SSSP), breadth first search (BFS) and Sparse matrix-vector multiplication (SPMV). Google search engines uses PageRank algorithm to list related searches. This algorithm calculates the popularity of the page. This popularity of the page is measured by number of pages, which links to it. When many pages have links to one page, it will be assumed that the page is more important. However one can artificially create new pages which link to his/her page to increase the popularity of his own page. To avoid this situation PageRank looks the popularity of other links, which linked to the page as well. The PageRank of different links can be calculated as:

$$PR(A) = (1 - d) + d((PR(Ti)/C(Ti)) + \ldots + (PR(Tn)/C(Tn))) \tag{1}$$

Where d = damping factor, PR = PageRank value, T = vertices have links to A, C = number of vertices Ti linked to.

The other common kind of algorithm is Breadth first search algorithm. This algorithm is used for transverse and search data, which is stored in tree data structure. It starts from source vertex and transverse all neighborhoods until all vertices of a graph are visited. Pseudo code of BFS algorithm is shown in figure 5.

The SSSP algorithm computes the shortest path from a source vertex to other vertices. It needs more memory space than BFS because each vertex may be visited more than once. The other algorithm called SPMV is used to calculate multiplication of a sparse matrix. A sparse matrix is simply a matrix with many 0 values.

```
1   Procedure BFS(G,S)
2   For each vertex v ∈ V do
3           visted[v] = false
4           value[v] = ∞
5   end for
6   // s is start vertex
7   visited[s] = true
8   value[s] = 0
9   add s to Q // Q is a queue
10  while Q is not empty do
11          u = pop(Q)
12          for each n neighborhood to u do
13                  if not visted[n] then
14                          visted[n] = true
15                          value[n] = value[u] + 1
16                          push(Q, n)
17                  end if
18          end for
19  end while
20  end procedure
```

Figure 5: Pseudocode of BFS Algorithm

2.7 Data Compression Algorithm

Huffman algorithm

Huffman is a loss less data compression and decompression algorithm. In Huffman coding the compression of the data is based on frequencies, which follows a concept called code tree. Huffman coding almost compress data by half. A code tree is a simple tree with value 0 and 1. The left transversal assigned to 0 and the right transversal assigned to 1. The example of code transversal is shown in figure 6.

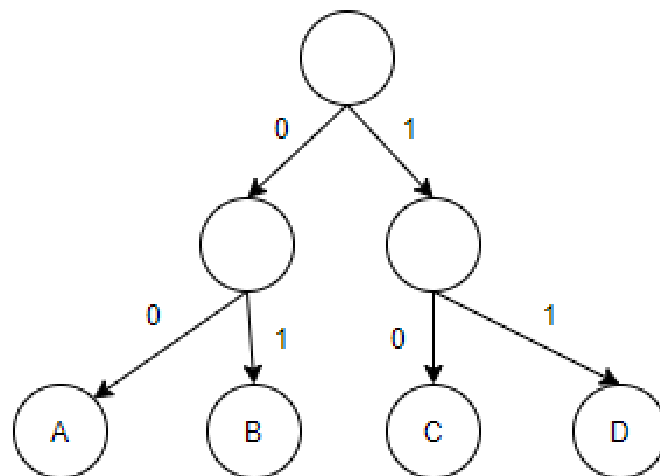


FIGURE 6: Code Tree

The main goal of Huffman coding is reducing the code tree and it works as follow.

Step 1: Create a leaf node for each node
Step 2: Add a leaf node to the queue
Step 3: While queue is more than one. Make a two leaf node tree from them. And make value of this node the sum of frequency of the two.
Step 3.1: Remove 2 nodes with low frequency
Step 3.2: Create node with the two leaf nodes. The value of the node is the sum of the two frequencies.
Step 3.3: Add the new node to the queue

Step 4: Only one node in the queue, the remaining node is the root node and the Huffman algorithm completes.

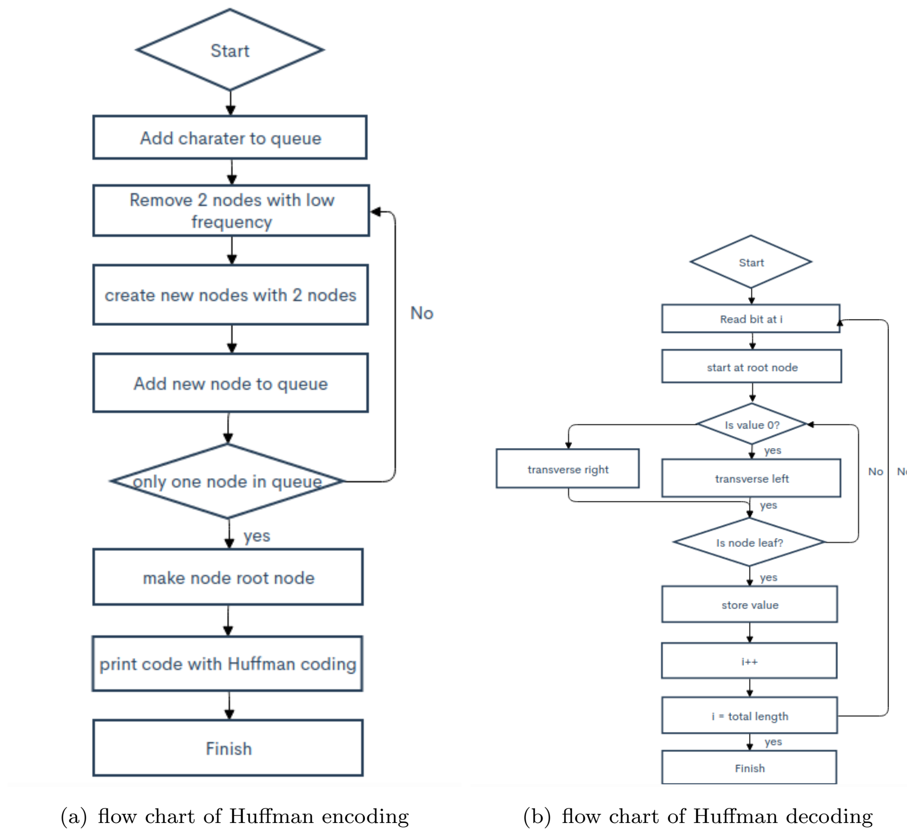
The Huffman encoding algorithm is shown below:

Huffman Encoding:

```
Algorithm Huffman(A):
  n = A.size()
  Q = priorityQueue()
  for i = 1 to n
    n = node(A[i])
    Q.push(n)
  end for
  while Q.size() != 1
    T = new node()
    T.left = x = Q.pop
    T.right = y = Q.pop
    T.frequency = x.frequency
    + y.frequency
    Q.push(T)
  end while
  Return Q
```

The decoding of the compressed data is simply changing stream of prefix value to individual byte value. It works as follow:

Step 1: for all compressed bit values
Step 1.1: start from root node
Step 1.2: while the node is not leaf node.
Step 1.2.1: if value is 0 transverse to left and if value is 1 transverse to write
Step 1.3: If leaf node print leaf node value.



(a) flow chart of Huffman encoding (b) flow chart of Huffman decoding

FIGURE 7: Huffman Algorithm flow chart

The flow chart of Huffman encoding and decoding are shown in figure 7.

The Huffman decoding algorithm is shown below:

Huffman Decoding:

```
Algorithm HuffmanDecoding(root, S):
   n := S.length()
   for i := 1 to n
      temp = root
     while temp.left != NULL
       and temp.right != NULL
       if S[i] = 0
         temp <= temp.left
```

```
        else
          temp <= temp.right
        endif
      i := i+1
      endwhile
      display temp.value
    endfor
```

Chapter 3

Literature Review

Given wide applicability of graph models, developing efficient graph processing accelerator architecture has been a great research interest area. Accelerators feature of massive parallelism and high memory access had attracted a lot of researchers to investigate how to apply accelerators to improve computation. However, Graph processing accelerators face such problems: (1) Poor locality, (2) high memory bandwidth requirement and (3) low memory capacity[17].

A graph algorithm has random memory access. Due to this property of graph algorithm the processor can't predict the next memory access, which is known as poor locality. The other challenge of graph computation is, a graph data has a high memory communication to computation ratio which leads to a high memory bandwidth requirement. In addition, most of graph data are large in size. Therefore providing memory which can store this large data is another issue. To overcome those challenges different research papers have proposed different solutions such as processing on CPU, processing on GPU, in memory processing and external memory support based processing[15,16,17].

3.1 Related work on Graph processing Hardware Architecture

3.1.1 Conventional System

Conventional approach which load data from memory to CPU and do computation is not suitable and efficient method for data intensive application [12]. The traditional memory hierarchy consists of storage to cache. The cache implementation is introduced to make data near to memory. However, graph computation has random access of memory that means the cache hit rate is very low. This means that the probability of finding a data from cache is very low. Abraham et al. (2018) tried to increase hit rate of cache based on the property of real world graph data. Most real world graph data follows power law distribution. That is 80% of edges are incident to approximately 20% of vertices. Using this property of graph they tried to store 20% of vertices in to the cache to increase the hit rate. They showed improvement over the conventional method.

The evolution of hardware architecture from early computers to in-memory computation is shown in figure 8 [12]. Early computers only have DRAM and processors computers. To speedup computation a single core with embedded cache was introduced. The cache is used to store data, which is used by core frequently.
To further improve performance of computers multi core with different level of the cache was introduced. This approach is used as a conventional architecture of current computers. However some computations need high memory communication. Therefore the conventional system fails for data intensive applications. There are two categories of graph processing in conventional system. These are distributed systems and servers. Distributed systems use many processors to support massive data. But this kind of approaches suffers from communication overhead and load imbalance issue. The other category called servers holds most of graph data in large main memory. These kind of systems are very rare because of high hardware cost. This high hardware cost is due to need of main memory, which has high memory capacity.
To improve the issue of conventional system near memory computing is introduced. In near memory computing some cores are stored near memory as shown in figure 8(d). To

further increase performance of computers in data intensive applications, approach called computation in memory introduced. In this approach the compute unit is in the memory. The detail of this approach is explained in the next section.

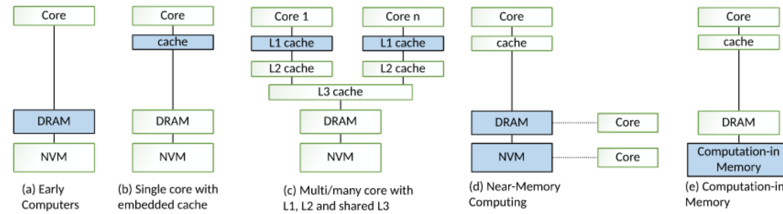


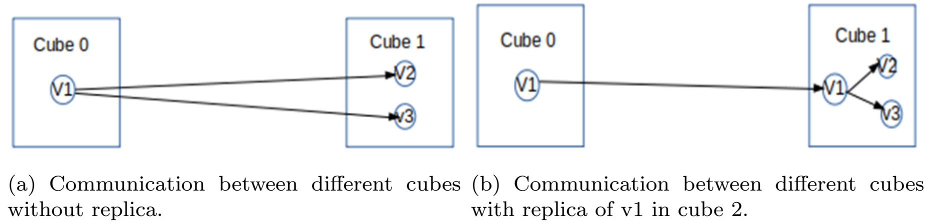
FIGURE 8: Evolution of hardware architecture [12].

3.1.2 Graph processing on GPU

GPU shows greater performance than conventional CPU systems but they still suffer from issues such as load imbalance and superfluous global memory access. Therefore graph applications don't work well in graph processing systems [13]. The other issue is GPU systems are known with relatively high energy consumption.
Amir et al. (2018) proposed a new graph transformation method called split transformation to overcome load imbalance issue. In split transformation approach, irregular graph data is converted in to more regular one by splitting the high degree value vertices. The degree of a graph V is the number of vertices adjacent to a vertex V. They test this approach in GPU processor and it showed improvement. However, in split transformation a high degree graphs are represented in two layers, which leads to more memory usage. In their proposed approach a graph represented as virtual layer and physical layer. This is to avoid a physical graph transformation. But this leads to additional memory usage to store both data. In addition this approach fails for HMC based accelerators. We tried to implement this approach in HMC and it reduced the performance of conventional HMC system [9]. These is because splitting of data leads to pressure on processing unit. But when we see HMC systems, they are designed for applications with low computation to memory communication ratio. That is they have relatively minimum processors than memory. Therefore splitting of data will add more pressures to these limited processors. Then this leads to worst performance in HMC based system. On the other hand for GPU system, it works efficiently. That is because GPUs are designed as multiprocessor units.

3.1.3 In memory processing

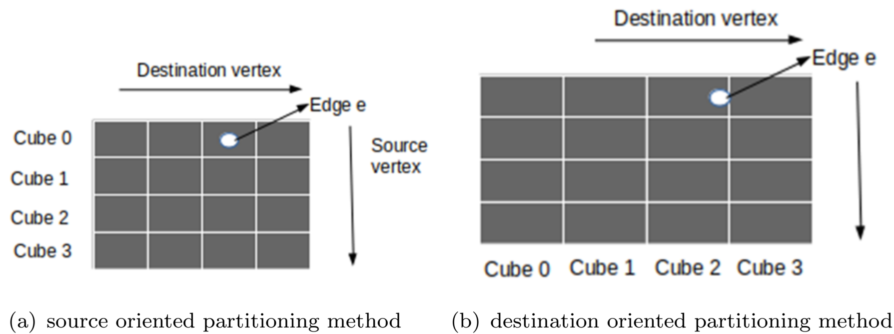
As explained before, conventional architecture is not able to perform computation efficiently for data intensive application. The emergence of this 3D memory technology attracted most researchers to focuses on in-memory processing to overcome graph computation issues. In this approach processing unit is on memory that is where the data resides. These kinds of approach have a great advantage for data intensive applications because they provide high bandwidth. These can be implemented by the emerged 3D technologies such as: HMC, High bandwidth memory (HBM) and ReRAM. This method reduces data movement and, thus, it solves high bandwidth requirement issue of graph processing. However it provides limited performance due to high communication overhead between inter memory cubes. This issue is explained by [9] that there is a need of additional work to utilize the bandwidth provided by those 3D technologies.
To overcome this issue [16] proposed efficient data partitioning mechanism. They proposed source cut partitioning mechanism to improve communication between different partitions of cubes. They partitioned edge data based on its destination vertex. Therefore, if edge(u,v) is assigned to one cube, all edges with the form (*,v) will also be assigned to the same cube. In addition each source vertex has replica in other cubes as shown in figure 9(b). In each update of source vertex the corresponding replica of the vertex in the other cubes will be updated. They called this updating process as replica synchronization. For example consider master vertex vi in cube 1 shown in figure 10. And let assume the only cubes which contains edges, which starts with vi, are cube 0 and cube 2. In replica synchronization, the cube containing vi needs to send updated value of vi to cube 0 and cube 2. In this case one massage, which contains vi value, will be sent to cube 0 and cube 2. Even if there are many edges starts from vi in cube 0 or cube 2, sending one message is enough for further processing. By this way improvement of intercommunication between cubes can been achieved. However, this kind of partitioning scheme has drawback such as use of additional replica leads to additional memory usage and graphs are difficult to partition to optimize network traffic, balancing load, etc... Song et al. proposed ReRAM based graph processing by using computation in compressed format concept. A sparse matrix can be represented as a compressed matrix representation. Most of previous systems



(a) Communication between different cubes without replica.

(b) Communication between different cubes with replica of v1 in cube 2.

FIGURE 9: Replica representation of vertices



(a) source oriented partitioning method

(b) destination oriented partitioning method

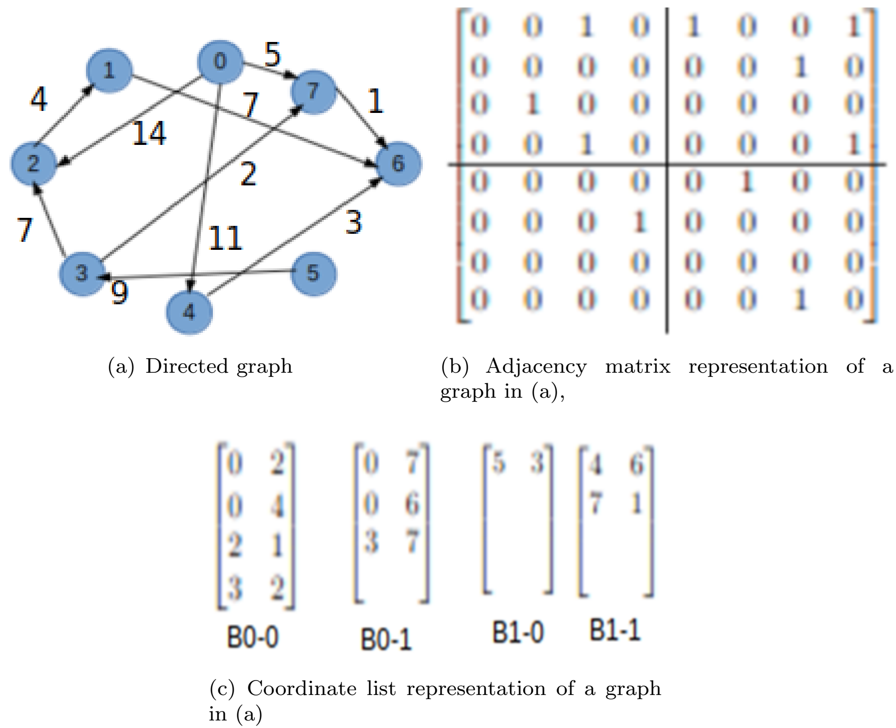
FIGURE 10: Graph partitioning methods

decompress compressed matrix before processing it. To further improve the hardware architecture of graph processing system, they used compressed format for further processing without decompressing it [17]. This has the advantage in three ways: graph computation to memory communication ratio can be increased, inter memory communication can be reduced and no need of additional unit to decompress data. The compressed matrix format of a sample graph is shown in figure 11 (c).

Leul et al. (2019) based on the observation of graph algorithm, they proposed in route computation concept. Most of graph computations are commutative by nature. Based on this observation they implemented computation unit in each router. This approach showed improvement over the baseline system [9].

3.1.4 External memory support based processing

Alternatively, disk-based single-machine graph processing systems stores active graph data in memory and store the remainder to disks [18]. This is because the real world graphs are so large that main memory can't hold. Disk based processing has slow speed because



(a) Directed graph

(b) Adjacency matrix representation of a graph in (a),

(c) Coordinate list representation of a graph in (a)

FIGURE 11: Graph Representation formats

there is high distance between processor and the disk. Therefore this kind of approaches can't be efficient for graph computations.

3.2 Related work on Graph compression techniques

The graph compression algorithms can be lossless and with loss. The graph data is very sensitive to loss, therefore a lossless compression algorithms are must in graph compression. Rushabh et al. (2018) Proposed graph compression method using Huffman coding. They applied this compression algorithm on graph data, which stored in memory, and showed up to 80% reduction in space required to store the graphs as compared to using the adjacency matrix.

Yuho et al. (2006) proposed a new packet data compression technique for network. From their custom compression algorithm they were able to compress a packet of five data in to a packet of three data. This technique showed 48% performance improvement over other network on chip architectures [25]. Cheng et al. (2018) proposed new approach

to store data and transfer data in compressed format in HMC based memory. They mainly considered to reduce communication between HMC and other system for general applications. Their proposed architecture showed improvement over baseline system by 42.4%. But still for graph computations, we can further improve performance by providing reduction of transmission of data between HMCs using reduction of packet sent in them. Our proposed system has used compression technique in transferring and storing data. In our baseline system data was sent between each HMC in packet format. By using compression technique to compress data, we were able to send two packets per time. Cheng et al. (2018) proposed new approach to store data and transfer data in compressed format in HMC based memory. They mainly considered to reduce communication between HMC and other system for general applications mot specifically for graph algorithm. Their proposed architecture showed improvement over baseline system by 42.4%. However they didn't consider the packet sent between HMCs. In our proposed approach we were able to add compress unit to reduce packets sent between HMCs and got impressive result.

Summary

Graphs are powerful abstraction mechanisms for representing relationships between data entities. It plays important role in different areas. In this section, a number of graph processing systems and the challenges inherent in processing graph applications are reviewed. Graph processing accelerators face (1) Poor locality, (2) high memory bandwidth requirement and (3) low memory capacity issue. To overcome these problems researchers proposed several ways to perform graph processing such as conventional system, GPU based system, in memory processing and external memory support. Conventional architecture and GPU based systems didn't work very well for graph applications. To overcome problems of this system different researchers did different works [19, 15]. However there is still a gap in performance of these systems due to a requirement of high memory communication in graph applications. To overcome this issue, in-memory processing approach is introduced. This approach provides efficient system for data intensive applications by reducing the distance between memories and processing unit, since the processor is stored within memory. However this kind of system faced high intercommunication between cores issue [9]. To overcome this problem different researches have been and on being done on

this area [16, 9]. But still efficient use of In memory processing hardware is an open question.

Chapter 4

Proposed Methodology

4.1 The baseline system

We selected Tesseract[9] and messageFusion[10] architec- ture, which uses 32 vaults in each HMC, as our baseline system. The logic layer of HMC in baseline system contains processing element. This processing element performs graph computations. And the system uses 16 HMCs interconnected by mesh topology as shown in figure 2. We selected mesh topology because of its efficiency. In mesh topology each node can transmit to and receive from more than one node at the same time.

In the baseline system the storage is divided in to three components such as Active list scratchpad, source scratchpad and destination scratchpad. The source scratchpad stores all vertices within a partition for current algorithm iteration. Destination scratchpad stores updated vertex information temporary. And the active list scratchpad tracks the active vertices in the iteration. The overall structure of the storage is shown in figure 12.

Our base line system has reduced unit in each routers to provide computation within a network. This approach showed 1.7x improvement from previous baseline architecture called Tesseract [9]. The overall architecture of our baseline system is shown in figure 13.

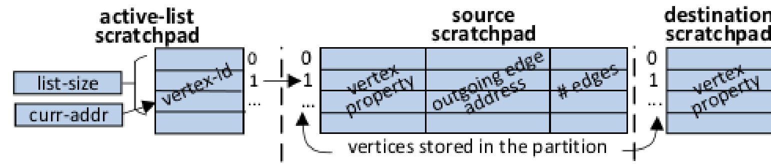


Figure 12: storage of graph data

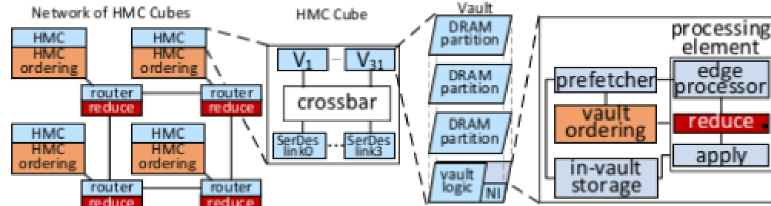


Figure 13: The baseline system architecture [10]

The baseline system consists of four HMCs interconnected by mesh topology. Each HMC connected with other by router. The router contains Reduce module, input buffers, VC allocator, switch allocator and crossbar switch. The router architecture of proposed system is shown in figure 14. Reduce module is used for in route computation. Reduce phase of a graph computation is handled by this module. The other components of a router are Input buffers, which used to store temporary data for computation. And a single buffer is assigned to each input.

The other element of router is VC allocator. It assigns virtual channel to each input. Virtual channels are buffer management flow control systems. The other component called Crossbar switch is used to connect inputs to outputs. For example in n x m crossbar switch, n inputs are connected to m outputs [20]. The typical cross bar switch is shown in figure 15.

In the HMCs network data is sent as packet from source to destination node. A packet is combination of flits. A flit consists of head flit, body flit and tail flit [20]. The body flit consists the data and the head flit consists the information required to transverse data.

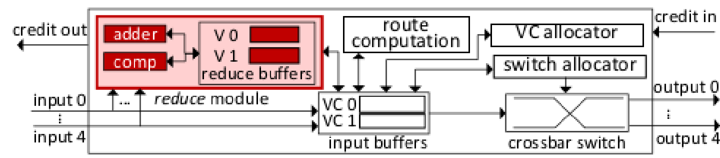


FIGURE 14: router architecture of proposed system

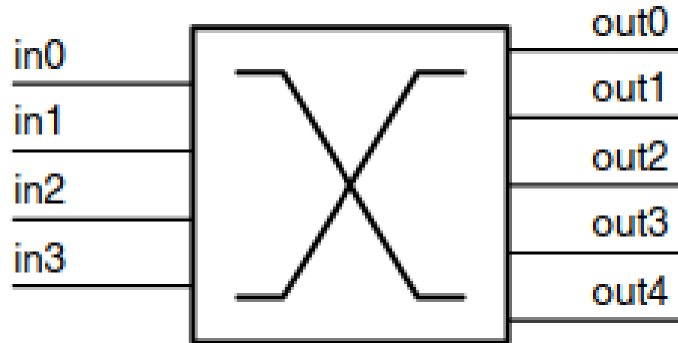


FIGURE 15: four inputs with five output crossbar switches[20]

4.2 Our proposed architecture

We added the hardware compressor and decompressor in baseline architecture. The compressor used to compress data before it sent in to the network. Our proposed architecture is shown in figure 16. The data from a router is in the form of array of integers. It contains source vertex id, destination vertex id, destination core id, and value of a vertex. Our proposed system works as follow. First the input data is stored in buffer. Then it pass through decompression unit. This decompression unit performs decompression to the input data. Then the decompressed data is inserted in to route computation unit. The route computation unit computes graph computation based on input data. Then the result will be compressed using compression unit. This compressed data will be sent to other router. Instead of sending one flit per packet we are able to send more than one flit per packet. This reduces total number of packet send between nodes and reduces the overall computational time of graph processing system.

In this research Huffman encoding algorithm are used to compress and decompress data. The detail background explanation of Huffman coding and cantor paring algorithm are

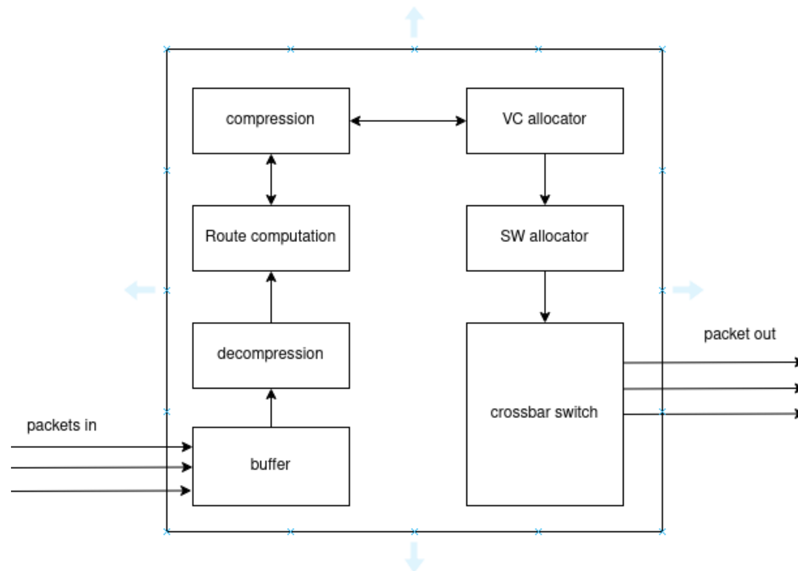


FIGURE 16: baseline system with helper(compressor)

found in section 1.2.5. we implemented Huffman encoding algorithm in c++ code. Efficient hardware implementation for Huffman algorithm is beyond this scope.

4.3 Hardware simulators

We used booksim to simulate interconnection between HMCs. Booksim [23] is a open source simulator used to develop inter connection networks. It supports 10 topologies. The simulator follows event driven simulation approach. It is developed in C++ programming language. And It helps to design route algorithms, allocators allocation and packet organization. However It doesn't support area and power simulation.We simulated the energy and power consumption of system using Orion[26]. Orion is power, area and energy estimation tool for network on chip architectures. It useful to model a power consumption for links, crossbar switches,and buffers. It is implemented by c++ programming language. We used CasHMC to simulate hardware architecture of HMCs. CasHMC is an open source simulator used to develop HMC arcitecture. It is developed in c++ programming language. The main advantage of this simulator, is it have object composition similar to actual HMC hardware[24]. CasHMC implements data request packets with various widths

ranging from 16 bytes to 256 bytes. Every character of HMC can be simulated using this software.

Chapter 5

Experiment

5.1 Experimental setup

We modeled the baseline system using a software called Booksim and CasHMC. The research used 16 HMCs which connected by mesh topology. The 4x4 mesh topology with interconnect of HMCs is modeled by Booksim and the internal structure of HMC is modeled by CasHMC. Each routers has four virtual circuits and each HMCs has 32 vaults. Each vaults provides 16 GB/s of internal memory bandwidth. We use conventional Huffman encoder/decoder to compress/decompress data. And we modeled the power and energy of proposed system using ORION 3. The hardware and software specification of the machine used for this thesis is given in table 1.

Specification of the machine used in this thesis	
Manufacturer	Dell Inc.
Processor	Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz
Memory	8GiB, SODIMM DDR3
Operating system	Linux 5.4.0-70-generic

Table 2: The hardware and software specification of the machine used in this thesis

5.2 Workloads

We used four common graph algorithms to check the performance of proposed architecture, which are Sparse Matrix-Vector Multiplication(SpMV) , PageRank(PR), Bredth first search algorithm(BFS), Single source shortest path algorithm(SSSP). We evaluated these algorithms in four data sets ranges from mid level to large data set. These data sets are from SNAP[21], which is shown in Table 2. In the table we report vertex and edge sizes of the data-sets.

5.3 Evaluations Result

5.3.1 Computational speed evaluation

Speed up obtained by our proposed architecture over baseline system is almost 1.7x in average.

Name	Description	vertices	edges
WV	Wiki-vote1	7.1K	103.7K
WG	Web google3	875K	5M
CA	RoadNet-CA4	2M	2.8M
SD	Slashdot2	82.2K	1M
OK	Orkut5	3.1M	117.2M

Table 3: Data sets used in our experiment [21]

From the total computational time of graph computation almost 60 % is from memory communication[10]. Therefore to see significant improvement on computational time, it is important to reduce communication between memories.
The improvement of speed in Page-rank algorithm is shown in figure 21. The peformance improvement on web-google data-set is 2x over baseline system Tesseract[9] and 1.7x over MessageFussion[10]. For lower size graph data such as Wiki-vote the improvement is relatively low. The improvement of computational time increase while the size of input graph data edges increases for PageRank algorithm. From this observation we can conclude that the proposed approach is effective on graph data-sets which have large edge count, for

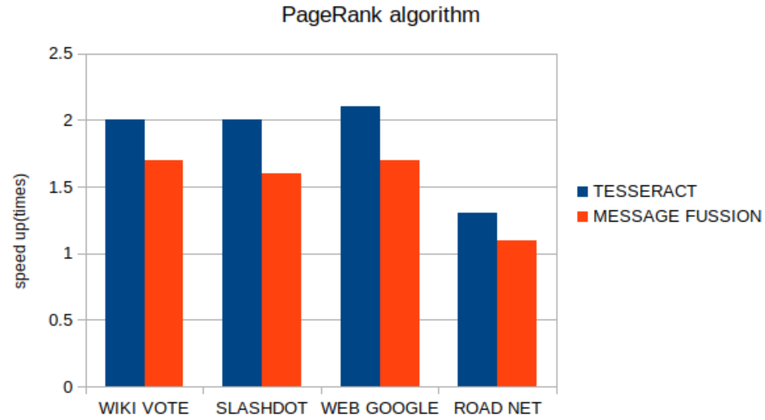


FIGURE 17: Performance improvement over Tesseract[9] and MessageFussion[10] for pagerank algorithm

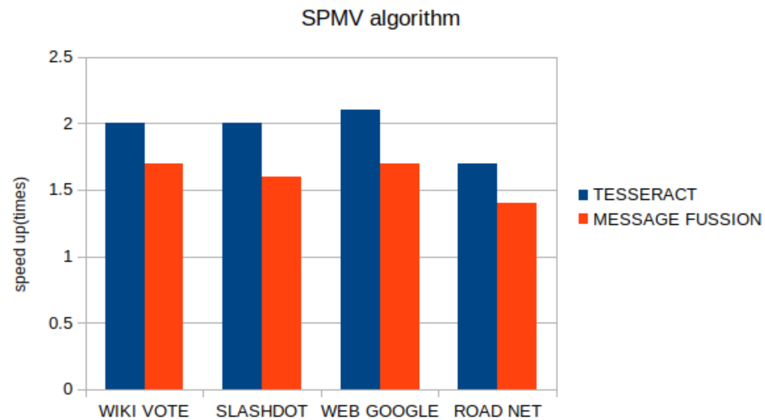


FIGURE 18: Performance improvement over Tesseract[9] and MessageFussion[10] for SPMV algorithm

PageRank algorithm. The improvement of speed in SPMV algorithm is shown in figure 22. The result obtained from this algorithm is simlar with PageRank algorithm. Again the improvement of speed for web-google data-set is 2x over baseline system Tesseract[9] and 1.7x over MessageFussion[10] as the same case for PageRank algorithm. The improvement of computational time increase while the size of input graph data edges increases for SPMV algorithm. From this observation we can conclude that the proposed approach is effective on graph data-sets which have large edge count, for SPMV algorithm. The similarity of the performance in PageRank and SPMV is because of algorithm similarity. These two algorithms are tested for unweighted graphs. The performance improvement of proposed

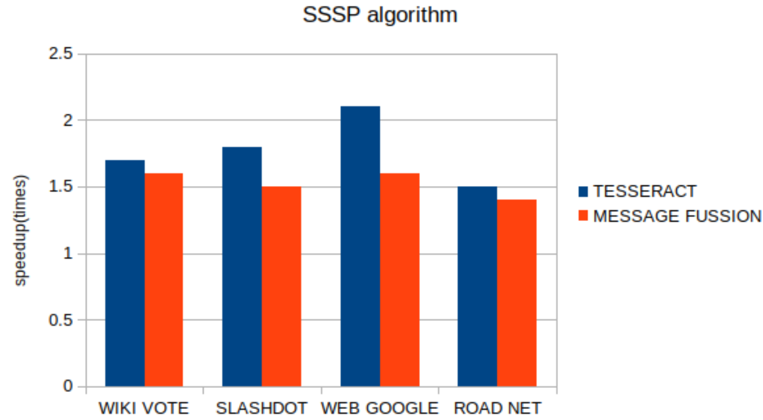


FIGURE 19: Performance improvement over Tesseract[9] and MessageFussion[10] for SSSP algorithm

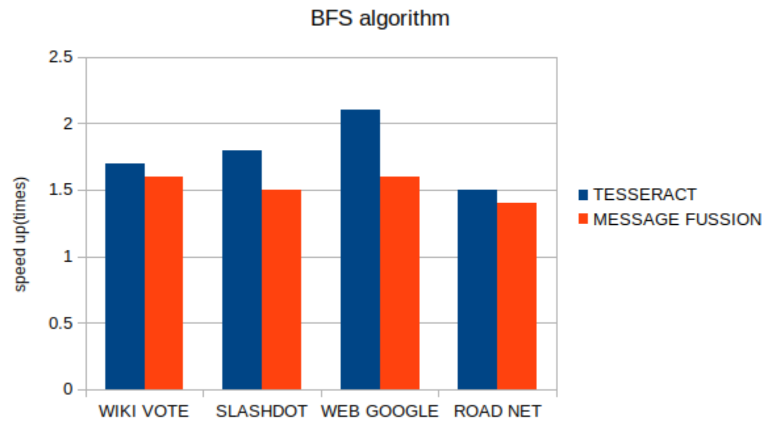


FIGURE 20: Performance improvement over Tesseract[9] and MessageFussion[10] for BFS algorithm

architecture over Tesseract[9] and MessageFussion[10] for SSSP algorithm is shown in figure 23. From observation the high improvement in proposed architecture is achieved in Web-google data-set. The performance improvement is about 2.2x over Tesseract[9] and 1.7x over messageFussion[10]. We used weighted graph version of the data-sets, because the algorithm works on weighted graphs.
In case of BFS algorithm the average improvement in the performance is 2x. Like other algorithm high achievement is gain for web-google data-sets. There is performance similarity between SSSP and BFS algorithm because of their algorithm similarity. Total number of messages sent in the network for pageRank algorithm is shown in figure 25. From observation there is high reduction of message traffic in our proposed algorithm over

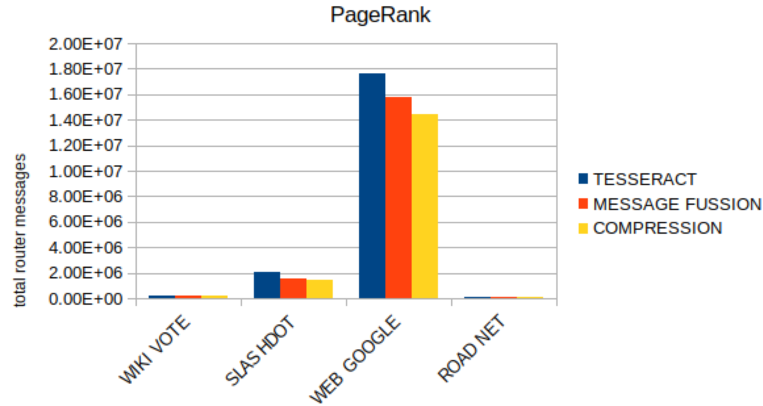


FIGURE 21: total number of messages in network for PageRank algorithm

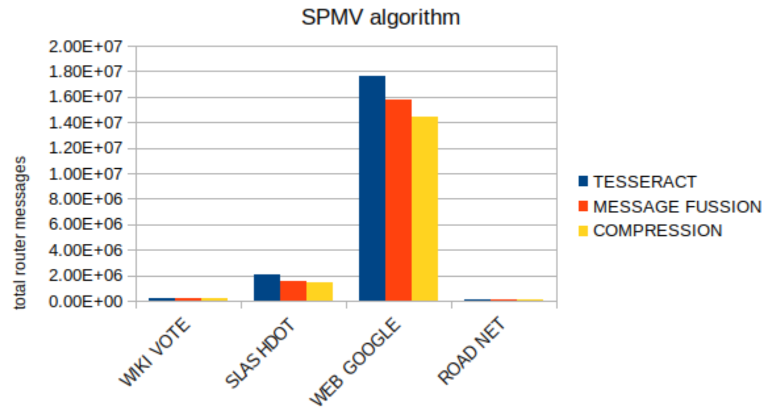


FIGURE 22: total number of messages in network for SPMV algorithm

the Tesseract and messageFussion. Web-google data-set achieved higher improvement than the other data-set in pageRank algorithm. This is due to it has the highest edges from the others. As edge increases the message between core increases. In Wiki-vote and Road-net data-sets our architecture showed similar performance with the other systems. From this we conclude that the high performance achieved in the higher edge count data-sets. The same condition is happened for SPMV algorithm.

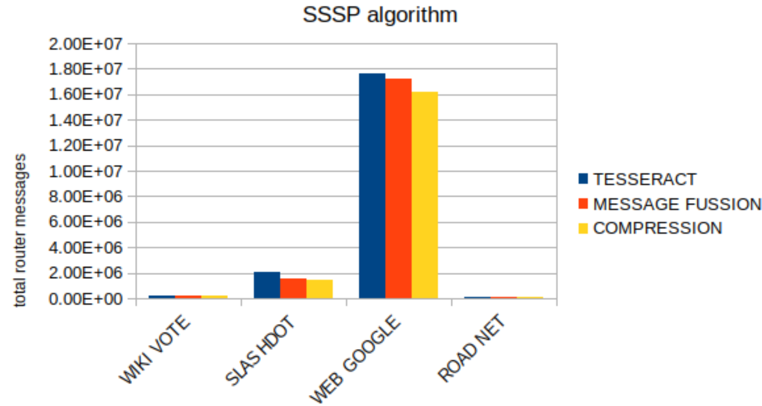


FIGURE 23: total number of messages in network for SSSP algorithm

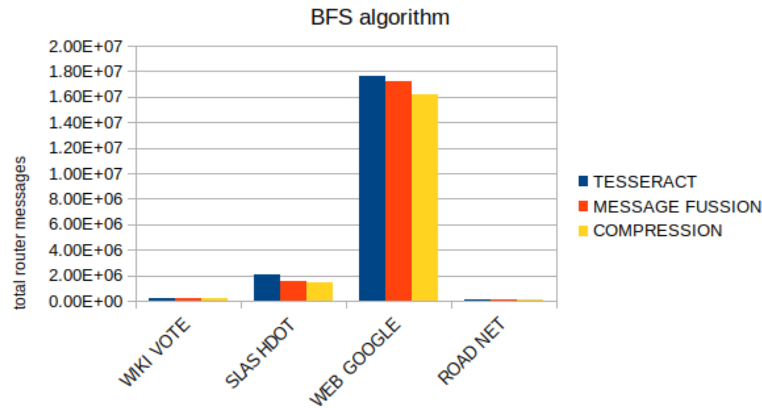


FIGURE 24: total number of messages in network for BFS algorithm

5.3.2 Energy and Area Evaluation

We evaluated power and energy consumption of the system using Orion 3. The energy consumption by transmission of the network is reduced by 47.28% over the baseline system. This is a high achievement because 62% of overall energy consumption is because of communication[10].

The baseline system took the total of 13.1% of the available area. And Huffman compressor/decompressor unit takes 25% of total area. Therefore our proposed system takes the 38% of available area. The area of this system can be reduced using efficient design of compressor and decompressor but this is beyond the scope of this thesis.

5.4 Future work

we recommend efficient design of compressor/decompressor unit as a future work. By efficient design of graph compressor/decompressor unit, a higher performance and the decrease in area of the system can be achieved.

Chapter 6

Conclusion

In conclusion, graph applications has great advantages in machine learning, social media application and so on. For speed improvement of graph applications, HMC based hardware is introduced. Even though this hardware can provides efficient bandwidth for graph processing, the intercommunication between memory parts of HMCs have limited the performance. To overcome this issue, We proposed compression unit in to it. Instead of sending plain data to another partition, our system compress the data before transmission and we reduce packet sent between HMCs by compressing packet data.This approach able to reduce number of data transmitted between memory partitions. We added our compression unit to baseline systems called MessageFussion [10]and Tesseract[9] . From the experimental result, our proposed approach showed 1.7x of performance improvement in average over our baseline systems. In addition the energy consumption by transmission of the network is reduced by 47.28% over the other two systems. Further improvement can be achieved by designing efficient compressor unit. Efficient hardware design of compressor/decompressor unit is beyond the scope of this thesis.

Appendix A

Appendix

Result in clock cycle for different algorithms and different hardware architecture

PAGERANK

	TESSERACT	MESSAGE FUSSION	COMPRESSION
WIKI VOTE	18565	16529	9761
SLASHDOT	286193	234821	144480
WEB GOOGLE	1832579	1440328	854889
ROAD NET	16249	13764	12455

SPMV

	TESSERACT	MESSAGE FUSSION	COMPRESSION
WIKI VOTE	18562	16399	9764
SLASHDOT	286084	234709	144110
WEB GOOGLE	1830829	1438581	851340
ROAD NET	16570	13718	9736

SSSP

	TESSERACT	MESSAGE FUSSION	COMPRESSION
WIKI VOTE	23721	22225	13555
SLASHDOT	327195	281019	182269
WEB GOOGLE	1840209	1285849	788933
ROAD NET	278542	304614	304174

BFS

	TESSERACT	MESSAGE FUSSION	COMPRESSION
WIKI VOTE	23721	22225	13555
SLASHDOT	327195	281019	182269
WEB GOOGLE	1840209	1285849	788933
ROAD NET	278542	304614	304174

Bibliography

[1] G. Vigna and R. A. Kemmerer. Netstat: a network-based intrusion detection approach. In *Proceedings 14th Annual Computer Security Applications Conference (Cat. No.98EX217)*, pages 25–34, 1998.

[2] Eugene Agichtein, Carlos Castillo, Debora Donato, Aristides Gionis, and Gilad Mishne. Finding high-quality content in social media. pages 183–194, 02 2008.

[3] Chris Biemann. Chinese whispers: An efficient graph clustering algorithm and its application to natural language processing problems. *Proceedings of TextGraphs*, pages 73–80, 07 2006.

[4] S. H. Corston, W. B. Dolan, L. H. Vanderwende, and L. Braden-Harder. System for processing textual inputs using natural language processing techniques. *May 31 2005, uS Patent*, 6901399, 2005.

[5] Rada Mihalcea and Dragomir Radev. *Graph-based Natural Language Processing and Information Retrieval.* Cambridge University Press, 2011.

[6] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[7] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

[8] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1–2):1–305, January 2008.

[9] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. *SIGARCH Comput. Archit. News*, 43(3S):105–117, June 2015.

[10] L. Belayneh, A. Addisie, and V. Bertacco. Messagefusion: On-path message coalescing for energy efficient and scalable graph analytics. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[11] Leul Belayneh and Valeria Bertacco. Graphvine: Exploiting multicast for scalable graph analytics. In *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, DATE '20, page 762–767, San Jose, CA, USA, 2020. EDA Consortium.

[12] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. Near-memory computing: Past, present, and future, 2019.

[13] Chuangyi Gui, Long Zheng, Bingsheng He, Cheng Liu, Xinyu Chen, Xiaofei Liao, and Hai Jin. A survey on graph processing accelerators: Challenges and opportunities, 2019.

[14] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*, pages 77–88, 2008.

[15] Abraham Addisie, Hiwot Kassa, Opeoluwa Matthews, and Valeria Bertacco. Heterogeneous memory subsystem for natural graph analytics. pages 134–145, 09 2018.

[16] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian. Graphp: Reducing communication for pim-based graph processing with efficient data partition. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 544–557, 2018.

[17] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen. Graphr: Accelerating graph processing using reram. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 531–543, 2018.

[18] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. X-stream: Edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, page 472–488, New York, NY, USA, 2013. Association for Computing Machinery.

[19] Amir Hossein Nodehi Sabet, Junqiao Qiu, and Zhijia Zhao. Tigr: Transforming irregular graphs for gpu-friendly graph processing. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, page 622–636, New York, NY, USA, 2018. Association for Computing Machinery.

[20] Amir Hossein Nodehi Sabet, Junqiao Qiu, and Zhijia Zhao. Tigr: Transforming irregular graphs for gpu-friendly graph processing. *SIGPLAN Not.*, 53(2):622–636, March 2018.

[21] William James Dally and Brian Patrick Towles. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[22] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[23] Hybrid Memory Cube Consortium. *Hybrid Memory Cube Specification 2.1*, 2014. Version. 2.1.

[24] Nan Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 86–96, 2013.

[25] D. Jeon and K. Chung. Cashmc: A cycle-accurate simulator for hybrid memory cube. *IEEE Computer Architecture Letters*, 16(1):10–13, 2017.

[26] Yuho Jin, K. H. Yum, and E. J. Kim. Adaptive data compression for high-performance low-power on-chip networks. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 354–363, 2008.

[27] A. B. Kahng, Bin Li, L. Peh, and K. Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *2009 Design, Automation Test in Europe Conference Exhibition*, pages 423–428, 2009.

[28] Heonchul Park, Jae-Chul Son, and Seong-Rae Cho. Area efficient fast huffman decoder for multimedia applications. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 3279–3281 vol.5, 1995.

[29] Marjan Karkooti. Huffman Encoder/Decoder. `https://www.ece.rice.edu/~marjan/vlsiproj/Huffman.htm`. [Online; accessed 19-July-2008].