Performance Evaluation and Optimization of HBM-Enabled GPU for Data-intensive Applications

Maohua Zhu*, Youwei Zhuo†, Chao Wang‡, Wenguang Chen§ and Yuan Xie*
*University of California, Santa Barbara, Email: {maohuazhu, yuanxie}@ece.ucsb.edu
†University of Southern California, Email: youweizh@usc.edu
‡University of Science and Technology of China, Email: cswang@ustc.edu.cn
§Tsinghua University, Email: cwg@tsinghua.edu.cn

***Abstract*—Graphics Processing Units (GPUs) are widely used to accelerate data-intensive applications. To improve the performance of data-intensive applications, higher GPU memory bandwidth is desirable. Traditional GDDR memories achieve higher bandwidth by increasing frequency, which leads to excessive power consumption. Recently, a new memory technology called high-bandwidth memory (HBM) based on 3D die-stacking technology has been used in the latest generation of GPUs, which can provide both high bandwidth and low power consumption with in-package stacked DRAM memory. However, the capacity of integrated in-packaged stacked memory is limited (e.g. only 4GB for the state-of-the-art HBM-enabled GPU, AMD Radeon Fury X). In this paper, we implement two representative data-intensive applications, convolutional neural network (CNN) and breadth-first search (BFS) on an HBM-enabled GPU to evaluate the improvement brought by the adoption of the HBM, and investigate techniques to fully unleash the benefits of such HBM-enabled GPU. Based on the evaluation results, we first propose a software pipeline to alleviate the capacity limitation of the HBM for CNN. We then design two programming techniques to improve the utilization of memory bandwidth for BFS application. Experiment results demonstrate that our pipelined CNN training achieves a 1.63x speedup on an HBM enabled GPU compared with the best high-performance GPU in market, and the two optimization techniques for the BFS algorithm make it at most 24.5x(9.8x and 2.5x for each technique, respectively) faster than conventional implementations.**

I. Introduction

Data-intensive applications process huge amount of data typically terabytes or petabytes in size [4]. Therefore, accelerating data-intensive applications has attracted many researchers from both academia and industry [2][3][5]. Neural network training and graph traversal are two of the most significant data-intensive applications used in data centers. For example, the prediction of attacking in network security systems is implemented based on neural networks and the rules are updated by graph traversal [11]. Therefore, there is a strong motivation to accelerate neural networks and graph traversal while reducing the computing power for data centers.

GPUs have been widely adopted in data centers to accelerate both neural network training and graph traversal for their high data throughput [3][12]. However, the downside of GPUs is its high power consumption comparing with other solutions such as FPGAs and ASICs. A large fraction of the power consumption of the GPUs is caused by their off-chip GDDR5 memories [13]. For the performance optimization of memory-bound data-intensive applications such as neural networks and graph traversal, improving the memory bandwidth by increasing the memory frequency will make the power consumption even worse. To boost memory bandwidth while addressing the power impact, AMD released the first generation of High Bandwidth Memory (HBM) [7] enabled GPUs. HBM is a new type of stacked DRAM memory that vertically integrates multiple memory dies. In the HBM-enabled AMD GPU, there are four stacks of memory chips sitting around the GPU chip, comprising of 4GB off-chip in-package global memory. The adoption of HBM not only increases the bandwidth of the device memory but also improves the power efficiency compared with traditional GDDR5 technology.

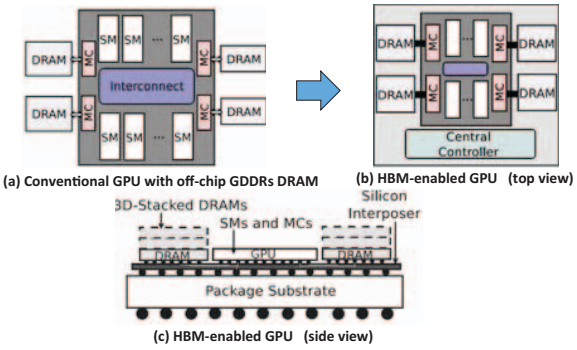

Fig. 1: The Architecture of HBM enabled GPU

However, the capacity of integrated in-package 3D memory is limited, due to technology and thermal challenges. Even though the industry is striving to integrate a much larger capacity of in-package memory in the future (for example, the future HBM2.0 standard may enable 32GB of in-package memory stacking [9]), currently, the users of the first generation HBM-enabled GPU have to address the capacity limitation for data-intensive applications. On the other hand, the number of compute units are dramatically increased on HBM enabled GPUs to exploit the wider width of the memory interface, which leads to severe load imbalance for applications with irregular memory access patterns. In conclusion, we need to solve the capacity bottleneck and the load imbalance issue of the HBM enabled GPU to unlock its high bandwidth benefit for the data-intensive applications.

In this paper, we evaluate the performance of an HBM enabled GPU comparing with state-of-the-art GPUs with GDDR5 memories for the cyber attack dectection task in a data center, in which (1) convolutional neural network training

This work was supported in part by DOE DE-SC0013553, NSF 1533933 and 1500848.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

and (2) breadth-first search are the most time-consuming algorithms. Based on evaluation results, we propose a software pipeline to reduce the memory usage to alleviate the memory capacity bottleneck for CNN training and two optimization techniques for better load balance of the BFS algorithm. Experiment results demonstrate that the HBM enabled GPU can achieve better performance than the baseline after applying our proposed techniques.

II. Evaluation and Optimization Methods

In this section, we first evaluate the two data-intensive applications on the HBM-enabled GPUs and then present our optimization methods to improve the performance for them, respectively.

A. Shrinking the Memory Footprint of CNN Training

1) Memory Capacity bottleneck: For CNN evaluation, we use Caffe [8] as our software platform. We first train the AlexNet [10] on the HBM enabled GPU. We observed that the AlexNet training is fast when the mini-batch size is 128. But for a larger mini-batch, for example, the default size 256, the network training becomes extremely slow. This is because the subscribed memory footprint of the network training is beyond the capacity of the HBM. OpenCL will automatically allocate memory buffers on CPU main memory if GPU has run out of the off-chip DRAM resource. The GPU reads and writes the buffers located on the CPU main memory via PCIE, of which the bandwidth is much smaller than the bandwidth of the HBM. To maintain the performance of the training, we should prevent the OpenCL runtime allocating OpenCL buffer objects on the CPU side by reducing the global memory usage.

2) Reducing the Global Memory Usage: The 4GB capacity of current HBM is not sufficient for the default AlexNet training, not to mention even larger networks such as VGG-16. In the original Caffe implementation, all memory buffer objects are allocated in the initialization phase and remain in the global memory during the whole training process. However, the layer structure of neural networks gives us a hint that not all neurons and connections are required by the compute units simultaneously. In fact, the data dependency in the back propagation algorithm requires only the output of one layer when processing the subsequent layer. Additionally, we observe that the training time of each convolution layer is no less than the time required to transfer the data block between CPU and GPU memory. Therefore, we can hide the data transfer with computation since the transfer is handled by the DMA. Thus we propose a software pipeline with a double-buffer to hide the memory transfer latency. In this approach, the memory prefetching, execution and data write-back of all layers are mapped to the same memory buffers respectively, which forms a three-stage pipeline with three buffers.

Figure 2 shows the time line of the workflow of the two command queues in OpenCL. When Layer i is being trained in compute units, the DMA copies the data of Layer $i-1$ from the GPU to the CPU and copies the data of Layer $i+1$ from the CPU to the GPU. At the time when the data of Layer $i+1$ is ready, Command Queue 2 generates an event to notify Command Queue 1 to continue training Layer $i+1$. At the end of the processing of Layer $i+1$, Command Queue 1 generates another event to inform Command Queue 2 to start data transfer. During runtime, the communication overhead is negligible. Therefore, the global memory requirement is reduced to the size of the double-buffer for the layers where the pipeline is applied. And as long as the computation time is longer than the communication time, the total training time will remain the same as the original implementation.

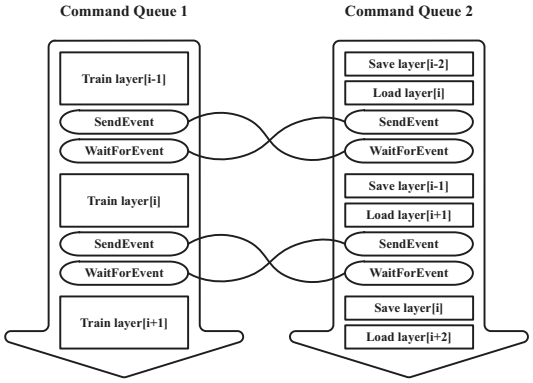

Fig. 2: Synchronization and Communication for Parallel Kernels in the Pipeline

B. Tackling the load imbalance issue in Breadth-First Search

As the HBM enabled GPU has higher global memory bandwidth, vendors have put more compute units on the chip to take advantage of the high bandwidth memory. However, the increase of computing resources makes the impact of the load imbalance issue even higher as more compute units will be idle. Therefore, the load imbalance has to be solved to make full use of the HBM enabled GPU.

Cycles	Warp 1		Warp 2		Warp 3		Warp 4	
	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
1	1->2							
2	1->3							
3	1->4							
4	2->1	3->1	4->1					
5	2->5		4->5					
6			4->6		IDLE	IDLE	IDLE	IDLE
7			4->7					
8			4->8					
9	5->2	6->4	7->4	8->4				
10	5->4		7->9					
11	9->7							

Fig. 3: The time line of the thread-centric scheme. The entries denote visits from the vertex on the left to the vertex on the right.

To deal with this load imbalance issue, we apply a warp-centric policy [6] instead of the traditional thread-centric implementation (3). The warp-centric task-thread mapping avoids divergence by assigning the search task of one vertex to one warp instead of one thread. One benefit from the warp-centric strategy is that the search task of one vertex can be parallelized. An additional trick is that after checking the vertex status in the Single Instruction Single Data (SISD) phase, the warp determines whether to continue processing in Single Instruction Multiple Threads (SIMT) or turn to the next vertex depending on the condition of the vertex status. The warp-centric searching process is illustrated in Figure 4.

Compared with the thread-centric scheme, the warp-centric reduces the total clock cycles by 36% in this case.

Cycles	Warp 1		Warp 2		Warp 3		Warp 4	
	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5	Thread 6	Thread 7	Thread 8
1	1->2	1->3						
2	1->4							
3	2->1	2->5	3->1		4->1	4->5		
4				IDLE	4->6	4->7		IDLE
5					4->8			
6	5->2	5->4	6->4		7->4	7->9	8->4	
7	9->7							

Fig. 4: The time line of the warp-centric scheme. The entries denote visits from the vertex on the left to the vertex on the right.

To further improve the performance of the BFS, we then exploit the idea proposed in [1] and design a bottom-up approach to traversing the graph on the HBM enabled GPU. It is an optimization especially advantageous for low-diameter scale-free graphs. The bottom-up BFS focuses on all unvisited vertices in lieu of dealing with the current frontier nodes. The bottom-up approach also traverses the adjacency list, but once in the list we find a neighbor in the current frontier, the traversal can stop in the middle and the unvisited vertex is identified present in the next frontier. For the bottom-up algorithm, it only has an upper bound-the neighbor edges of all currently unvisited vertices. Fortunately, the actual amount of access is expected to be much smaller, with many skipped queries. The bottom-up significantly reduces the randomness in memory accesses and thus achieves higher bandwidth on the HBM.

III. Experimental Results

In this section, we first describe the evaluation setup and then present the performance results of the proposed methods.

A. Experimental Methodology

Our experimental platform is a workstatioin with a six-core Intel Xeon E5-2603 CPU with 32GB main memory, and an AMD Radeon Fury X GPU. We also choose another two GPUs as the performance baseline: (1) AMD FirePro W7100 employs the same architecture as the Fury X GPU but it uses GDDR5 as the off-chip memory, and (2) NVIDIA Tesla K40 is the latest high-end GPU for general purpose computing in market. W7100 can be seen as a Fury X GPU without the HBM. K40 stands for the contemporary high-performance GPGPU, which is used to demonstrate how much benefit we can get from the HBM enabled GPU. The operating system is CentOS 6.5 and the compiler we use is GCC 4.4.7. The machine is installed with AMD OpenCL SDK 3.0 and CUDA 7.5.

B. Shrinking the memory footprint of CNNs

Figure 5 shows the execution time of forward propagation and backward propagation on different GPUs, respectively. We can observe that the straightforward implementation of CNN forward propagation and backward propagation on Fury X suffers significant performance drop. Even comparing with the W7100, which has fewer compute units and the same architecture as the Fury X, the performance is much lower. We further studied the underlying reason of this abnormal performance and found that the total memory requirement is beyond 4GB for AlexNet training. OpenCL automatically allocates memory buffer objects on the CPU host memory when the system is out of GPU device memory. Therefore, the compute units has to stall to fetch data from the CPU-end memory objects, which results in extremely low occupancy of GPU compute units. For HBM-enabled memory the situation is even worse because the large number of compute units on chip will cause severe PCIe congestion, which in turn downgrades the overall performance. To eliminate the long-latency data fetching, we applied the pipeline technique described in Section II-A2 to all convolution layers. One thing worth mentioning is that the pipelining is designed for shrinking the memory usage rather than improving the performance. By doing this, the total data requirement is reduced from 4.35GB to 3.52GB. After reducing the global memory usage, Fury X outperforms K40 with a speedup of 1.63x.

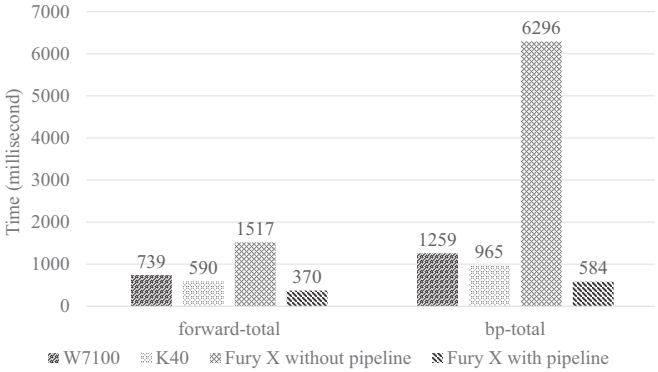

Fig. 5: Training Time of one Iteration

Although the data swapping overhead is carefully covered by computation, we found that the pipelined training is still 18% slower (mini-batch size: 256) compared with the un-pipelined implementation on the W7100 GPU, which has enough device memory space to hold all data and parameters on chip. Note that the pipelining scheme is not designed for the ideal situation that the device memory is large enough to hold all memory buffer objects. As our goal is to reduce the memory usage to eliminate the direct access from the GPU to CPU-end memory, we did not use aggressive strategies to minimize the memory usage in practise. For example, the computation of fully connected layers is simpler than convolution layers and thus it cannot completely hide the memory transfer. So we did not apply the pipeline to higher fully connected layers, since it can alleviate the performance overhead while keeping the memory usage less than the size of the device memory space.

C. Breadth-First Search Optimization

We evaluate the performance of the proposed methods for the BFS optimization on Graph 500 benchmark. The evaluation metric is the number of edges traversed per second, which is usually referred to Traversed Edges Per Second (TEPS). The TEPS metric is proportional to the throughput of the BFS algorithm. Input graphs are randomly generated by Graph500 implementation.

We first evaluate the performance of our warp-centric implementation. Although the warp-centric method maps one single

vertex to a warp, the warp size here is not necessarily the same as the hardware configuration (64 for AMD GPUs and 32 for NVIDIA GPUs) when it comes to implementation. As one vertex may have less than 64 neighbors, mapping it to one single hardware warp will result in resource under-utilization. Meanwhile, mapping more than one vertex to one hardware warp will lead to divergence and workload imbalance. In the first phase of evaluation, we swept the number of vertices mapped into one hardware warp and found out that mapping two vertices achieves the best performance.

Experiments show that, without the warp-centric method, the Fury X and the W7100 have no distinguishable difference in performance. Although the underlying HBM provides higher bandwidth, BFS cannot take advantage of it because of uncoalesced access pattern. However, after warp mapping, the Fury X achieves a speedup from 6.4x to 9.8x. This technique works as well for the W7100, but with a less significant speedup. Finally, the Fury X outperforms the W7100 by 1.5 times faster. This result demonstrates that the load imbalance issue has a higher impact on the HBM enabled GPU than traditional GPUs with GDDR5, and that our warp-centric method can unlock the benefit brought by the high bandwidth memory.

Based on the warp-centric implementation, we build another traversal program with the bottom-up strategy described in Section II. For example, Level 0 means the search from the source vertex and Level 1 means the search from the vertices visited by Level 0. For Level 0 and Level 1, the bottom-up implementations outperform the top-down solutions because the frontiers in these two layers connect to a large proportion of the graph and thus the top-down implementation has to traverse many edges which are invalid for the next round. While in such scenarios, the bottom-up scheme proves to be efficient because the skip condition is frequently satisfied.

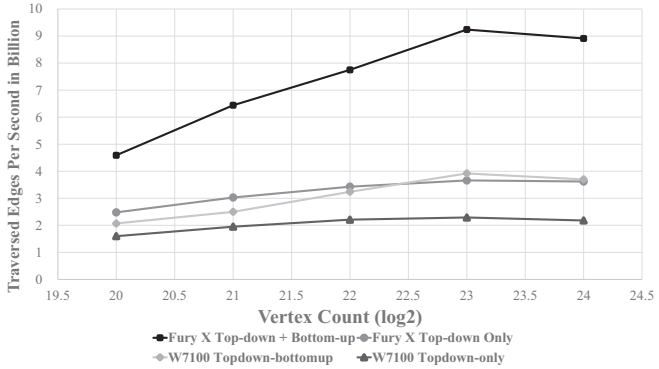

Fig. 6: Performance of Mixed Direction Approach

At last, by carefully switching directions between the top-down and the bottom-up schemes in different search levels, we can obtain the minimum execution time for each iteration, as shown in Figure 6. For the Fury X, the overall performance is 2.5 times faster than the previous top-down only implementation with warp-centric optimization. On the contrary, the W7100 benefits less from it, while speedup is only 1.7. The result implies a conclusion similar to that in Section II, that is, we have to tackle the load imbalance issue to make full use of the HBM enabled GPU. The evaluation results also suggest GPU architects introduce thread renaming and warp-level early exit mechanisms to further exploit the high bandwidth memory.

IV. Conclusions

HBM-enabled GPUs offer high memory bandwidth and low power consumption. However, the memory capacity that can be integrated inside the package is limited. Therefore, extra software techniques are required to efficiently implement neural networks and graph algorithms by exploiting the memory bandwidth benefit and address the capacity bottleneck. In this paper, we implement AlexNet and BFS on an HBM-enabled GPU to evaluate the performance gain brought by the in-package 3D stacking memory. The evaluation results show that AlexNet training encounters lack of memory capacity and BFS suffers low utilization of memory bandwidth and load imbalance. We then propose a software pipeline to reduce the memory usage of AlexNet training and apply two techniques, warp-centric method and search direction optimization for BFS. Experiment results demonstrate that our pipelined CNN training achieves a 1.63x speedup on an HBM enabled GPU compared with the best high-performance GPU in market. For the BFS algorithm, the warp-centric design achieves a speedup from 6.4x to 9.8x over the traditional thread-centric implementation, and the direction optimization provides an extra 2.5x speedup based on the warp-centric implementation.

References

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. *Scientific Programming*, 21(3-4):137–148, 2013.

[2] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. DianNao. *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems - ASPLOS '14*, pages 269–284, 2014.

[3] A. Coates, B. Huval, T. Wang, D. Wu, and A. Y. Ng. Deep learning with COTS HPC systems. *Proceedings of The 30th International Conference on Machine Learning*, pages 1337–1345, 2013.

[4] M. Gokhale, J. Cohen, A. Yoo, W. M. Miller, A. Jacob, C. Ulmer, and R. Pearce. Hardware Technologies for High-Performance Data-Intensive Computing. *Computer*, 41(4):60–68, 2008.

[5] S. Hong, S. K. Kim, T. Oguntebi, and K. Olukotun. Accelerating cuda graph algorithms at maximum warp. In *ACM SIGPLAN Notices*, volume 46, pages 267–276. ACM, 2011.

[6] S. Hong, T. Oguntebi, and K. Olukotun. Efficient parallel graph exploration on multi-core cpu and gpu. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 78–88. IEEE, 2011.

[7] A. M. D. Inc. High-bandwidth memory (hbm): Reinventing memory technology. *Scientific Programming*, 21(3-4):137–148, 2013.

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *Arxiv*, 2014.

[9] N. Kirsch. Nvidia pascal gpu with hbm2 taped out - gp100. *Legit Reviews*, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] L. Lu, R. Safavi-Naini, M. Hagenbuchner, W. Susilo, J. Horton, S. L. Yong, and A. C. Tsoi. Ranking attack graphs with graph neural networks. *Information Security Practice and Experience*, pages 345–359, 2009.

[12] D. Merrill, M. Garland, and A. Grimshaw. High Performance and Scalable GPU Graph Traversal. *Technical Report*, pages 1–15, 2011.

[13] J. Zhao, G. Sun, G. H. Loh, and Y. Xie. Optimizing GPU energy efficiency with 3D die-stacking graphics memory and reconfigurable memory interface. *ACM Transactions on Architecture and Code Optimization*, 10(4):1–25, 2013.