A Model for Scalable and Balanced Accelerators for Graph Processing

Marjan Fariborz, Mahyar Samani, Terry O'Neill
Jason Lowe-Power, S.J. Ben Yoo, and Venkatesh Akella

Abstract—Designing a graph processing system that can scale to graph sizes that are orders of magnitude larger than what is possible on a single accelerator requires a careful codesign of accelerator memory bandwidth and capacity, the interconnect bandwidth between accelerators, and the overall system architecture. We present a high-level bottleneck-analysis model for design and evaluation of *scalable* and balanced accelerators for graph processing. We show several applications of this model including how to choose the right mix of different memory types, network topology, network bisection bandwidth, and system-level architecture to match the access patterns and capacity requirements of different data structures for a given graph and a performance target.

Index Terms—Graph accelerator, graph analytics, heterogeneous memory, balanced computing

1 Introduction

Graph analytics is used to uncover insights from high volumes of connected data. As the size of real-world graphs increases, future computing systems must support large-scale graph processing in a cost-effective and timely manner. Today's largest benchmark graphs are on the order of 100 billion edges and 3 billion vertices [3], and some systems are processing graphs with over 200 trillion edges [6]. In addition to this enormous scale, graph processing has many challenges, such as frequent random memory accesses and low spatial and temporal locality [8]. These challenges are compelling a resurgence of studies focusing on implementing accelerators for graph workloads. However, due to the infeasible runtimes—and even more infeasible simulation times—these studies do not evaluate their performance at these massive scales. Thus, there is a *need* for an analytical model to predict the performance of graph processing as we scale the graph size (by two or three orders or more) to help design balanced high performance graph accelerators in the future. The goal of this paper is to establish this analytical model.

Most prior accelerators for graph algorithms focus on improving performance by reducing data movement and data accesses through on-chip memories [7], [8], [11] or perform processing in/near memory [4]. To scale to large graphs, these accelerators either use time multiplexing (temporal slicing), which is not scalable with memory capacity and does not improve performance, or they use a "scale-out" approach (spatial slicing) and duplicate their accelerators many times to make a large-scale system [7]. In the model presented in this paper, we focus only on spatial slicing, or scale-out approaches, since these are more scalable to massive graph analytics than temporal slicing.

Simply increasing the number of accelerators in the system does not guarantee performance improvement. The model presented in this paper allows us to investigate how to build a balanced large-scale graph processing system without simulation or emulation. The model parameters are specific to a particular system design, algorithms, and graphs of interest. To build the model, we ask the following questions. What data structures does the accelerator or algorithm use? What is the capacity requirement of the memory devices to store each data structure for the graph of interest? How does the access pattern of each data structure and bandwidth of the memory devices affect the performance of the algorithm? What are the network requirements (port bandwidth, bisection bandwidth, and topology) to fully utilize all memories and accelerators in the scale-out system?

With the answers to these questions, we can parameterize a model to find a balanced design which gives the most performance for the least cost (i.e., with the least over provisioning of bandwidth and capacity). Our model helps designers understand the performance implications of scaling out their accelerator cores to accommodate large graphs, and encourages designers to consider the impact of the whole system to create a balanced graph accelerator. Since well-known microarchitectural techniques like parallelism and better partitioning and exploiting data reuse via on-chip memory can be used to hide the memory latency, our proposed model mainly focuses on the systems' throughput.

The main contributions of this work are:

- We develop an analytical method to predict the performance of a graph processing accelerator taking into account memory and network performance and accelerator-specific optimizations such as degree of reuse and tiling schemes.
- We demonstrate how to take advantage of heterogeneous memory to address the unique access patterns and capacity/bandwidth trade offs in graph processing to develop a balanced system.
- We show three important applications of our model to show its applicability to computer architects and

- *The authors are with the Department of Electrical and computer Engineering and the Department Computer Sciences, University of California Davis, Davis, CA 95161 .*
- *Emails: {mfariborz,msamani,toneill,jlowepower,sbyoo,akella}@ucdavis.edu*

- *This work was supported in part by ARO W911NF1910470.*

system designers interested in optimizing large scale graph processing.

2 A Model for Scalable Accelerator

The proposed performance model calculates the system-level requirements assuming an asynchronous implementation of the widely used vertex-centric programming paradigm for graph processing. We assume a system is constructed by combining a set of accelerator tiles (the basic building blocks) with an appropriate interconnection network. The requirements are in terms of *memory capacity* and *memory bandwidth* for different types of memory, the *network bandwidth*, and *number of tiles*. These requirements depend on the structure of the graph, representation of different data structures, and physical constraints such as the maximum I/O on each tile. With the assumption that graph algorithms are throughput limited, these requirements capture the behavior of many graph accelerators. Additional constraints could be added to the model if this assumption does not hold (e.g., the compute capability is a limiting factor).

To calculate the bandwidth for each component, we consider the maximum performance required from each component individually. Equation 1 shows Graph Algorithm Iron Law (GAIL) [5] which calculates the execution time for graph workloads. GAIL separates algorithm and hardware performance. Traversed Edge Per Second (TEPS) shows the hardware performance. For the same algorithm implementation, larger TEPS will result in a smaller execution time. Our model focuses on maximizing this TEPS performance metric.

$$\frac{\text{time}}{\text{kernel}} = \frac{\text{number of edges}}{\text{kernel}} \times \frac{1}{\text{TEPS}} \tag{1}$$

2.1 Memory System

Next we describe what data structures to store in the memory and the capacity and bandwidth requirement of these data structures. Both edges and vertices have different characteristics. The proposed model calculates the capacity and bandwidth requirements of edges and vertices separately. For simplicity we only consider vertex and edge data structures in the description below, but the model can extend with other data structures easily.

2.1.1 *Edge*

In general, edges require a larger memory capacity compared to vertices (graphs such as WDC12, Twitter, and LiveJournal have 36×, 34×, and 15× more edges than vertices). In most graph programming models [1], [2], [9] access to edges are sequential and read-only. The model should determine the number of memory devices that provides enough capacity for a targeted graph. Table 1 Row 1 shows the required capacity for the edge memory which depends on the number of bits representing edge information. The size of the edge is different between accelerators. At a minimum it should include the vertex ID (destination or source) and the weight of the edge. The other constraint for the edge memory is the required bandwidth to achieve a certain performance in TEPS (shown in Table 1 Row 2).

2.1.2 *Vertex*

In contrast to edges, vertices have low spatial and temporal locality in most graph algorithms. This lack of locality causes inefficient off-chip memory access and low memory access throughput. Prior work exploits locality through graph pre-processing [8] or by online traversal scheduling [10] to improve the on-chip cache usage. Other hardware accelerators use on-chip SRAMs to store vertices and/or events to reduce off-chip memory access [11].

The required vertex bandwidth depends on the performance of the edge memory. For every edge read there is at most one vertex read and one vertex update. Therefore, vertex access rate is 2× higher than edge access rate. The maximum supported bandwidth also depends on the access granularity to the vertex memory system (a.k.a. *atom size*). On-chip caches reduce the off-chip vertex memory access rate. Therefore in our model we use a parameter α to model accelerators with on-chip memory assigned to vertices. α indicates the fraction of the off-chip memory bandwidth that is needed by the accelerator. α is a value between zero and one. An accelerator that exploits locality/reuse will have a smaller α which means it will have lower off-chip memory bandwidth requirement. Table 1 Row 3, shows the maximum required bandwidth for vertex memory given a TEPS (usually from the peak edge bandwidth (Table 1 Row 2). In addition to the bandwidth, we must also meet the capacity requirement of the vertex memory. Our model needs to consider the capacity of vertices (Table 1 Row 4).

2.2 Network Requirements

After finding the best memory technology for vertex and edge data structures, we must ensure that data movement among accelerators will not cause a performance bottleneck. We consider the *network* as the third constraint in our model. What we consider as network in our scaled-out system is the interconnection fabric between accelerators. Accelerators use this network to communicate inter-slice updates. A well-partitioned graph with a low inter-slice event rate does not require a high bandwidth network. The proposed model takes into account the bisection bandwidth, port bandwidth, and topology of interconnection network.

2.2.1 *Bisection Bandwidth*

The required bisection bandwidth is dependent on the performance of each accelerator and the size of the message communicated through the network. With every edge read a new message is created to another vertex. Hence, TEPS indicate the maximum rate of messages generated from the aggregate edge memory across all accelerators. We use γ to indicate the fraction of the messages targeting vertices in remote accelerators and have to be communicated over the network interconnect. Table 1 Row 5 shows the required network bisection bandwidth.

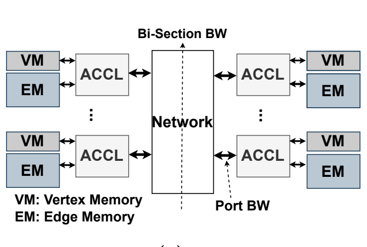
Data representation dictates the size of the message communicated in the system. It depends on the implementation of the accelerator and is a parameter in our model.

2.2.2 *Port Bandwidth*

The required network bandwidth for each accelerator can also be a limiting factor. Port bandwidth depends on the required bisection bandwidth and the number of accelerators in the system. See Table 1 Row 6.

2.2.3 *System Architecture*

The system architecture dictates the interconnection traffic patterns and depends on the port bandwidth and the in-



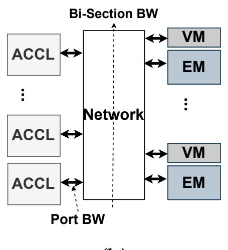


Fig. 1: System architecture for the scaled-out accelerator. a) Near memory processing. b) disaggregated memory-based.

1	Edge Capacity = Number of edges × Sizeof(Edge)
2	Edge Bandwidth = Maximum TEPS × Sizeof(Edge)
3	Vertex Bandwidth = 2× atom size × TEPS ×α
4	Vertex Capacity = Number of vertices × Sizeof(Vertex)
5	Bisection BW (Near-memory) = TEPS × Sizeof(message) ×γ
6	Port Bandwidth = $\frac{\text{Bisection Bandwidth}}{\text{Number of accelerators}}$
7	Bisection BW (disagg.) = Vertex BW + Edge BW

TABLE 1: System constraints used in the proposed model. α is the miss ratio of vertex on-chip memory. γ is the percentage of inter-accelerator to intra-accelerator communication.

terconnection network topology. We model two systems: near memory processing architectures and disaggregated memory-based architectures as shown in Figure 1. In the near-memory processing system, each accelerator tile is connected to a local memory system, and the interconnection network is between the accelerators. Here, each accelerator is operating on its local vertex and edge information, and only events are communicated through the interconnect (Table 1 Row 5). Whereas in the disaggregated memory based system the accelerators do not communicate directly with each other—the communication is through the memory. In this topology, accelerators not only use interconnects for communicating events but also use them for read/write vertex and edge information. Hence, it required a larger bisection bandwidth in the network (Table 1 Row 7).

2.3 Accelerator Node

The internal architectural parameters of accelerators are another constraints in our model. These parameters are: 1) *Number of bits* for representing data structures. This parameter impacts the bisection and port bandwidth of the network and also the edge memory bandwidth requirement. 2) *On-chip resources* such as caches and SRAMs. These on-chip memory systems will impact the required bandwidth from the main memory (α). 3) *The number of I/O pins* connected to each accelerator and their *data rate* limits the number memory nodes connected to the accelerators (total capacity supported by a single accelerator), and the port bandwidth of the interconnect. Given the high memory access and communication to computation ratios of graph workloads, we do not consider the computation within the accelerator as a bottleneck in our model [8].

3 Applications of Model

We show three concrete use cases (or applications) for the proposed model. In the first use case, we show how to use the model to *create* a balanced system taking advantage of a heterogeneous memory such as HBM3, Intel® Optane™, and DDR5 for a given graph. Next, we show how given a homogeneous memory system, the proposed model can be used to calculate the locality required from on-chip memory to store vertices and edges in the same memory. This could be useful to someone developing the microarchitecture of the accelerator. In the third use case, we compare the system generated from our model with a high performance computing system given a graph, algorithm, and performance.

In the first two applications, we use the WDC12 hyperlink graph [3]. We use 8 bytes to represent edges and 16 bytes for vertices. The accelerator for WDC12 requires 55 GiB for vertices and 2 TiB of edges. We consider the Breath First Search (BFS) algorithm and assume our accelerators have no on-chip resources i.e., $\alpha = 1$. In our partitioning scheme we assume 80% more inter-slice edges than intra-slice (γ). We assume our accelerators have 2000 data pins. Note, that these are just assumptions to illustrate the specific applications. The proposed model is not restricted to these assumptions.

3.1 Use Case 1: Creating a Scaled-out System

In this section, we address the question *How to create a balanced Scale-Out graph accelerator given graph input and a given performance target?* The proposed model provides the required memory devices for vertex and edge memory and network bisection bandwidth to answer this question.

Edge Memory: Figure 2a shows the capacity-performance relationship for different memory technologies using Table 1 Row 1 and Row 2. The right-side of the red dotted line shows the region of interest for edge memory capacity. HBM3.0 requires more than 128 stacks to scale to WDC12 graph, and although it provides a significant amount of performance, it also has high cost. Optane, provides a significantly larger capacity with 32× fewer channels than HBM3.0 stacks, but with less performance due to its lower read bandwidth. Given DDR5's improved cost-performance tradeoff, we choose its performance (100 GTEPS) as the target accelerator performance for the bottleneck analysis for the rest of the system.

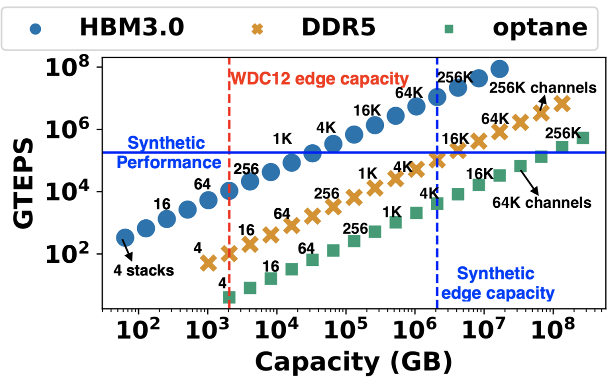
Vertex Memory: Figure 2b shows the performance-capacity trade-off for the vertex memory. The red dashed and solid red lines show the minimum capacity and performance requirements respectively. The upper quadrant region of these lines shows the memory systems with both performance and capacity considerations.

Network: For our network constraints we consider near memory processing system architecture due to its low bisection bandwidth requirements. With 8-byte message size and using the equation in Table 1 Row 5, the required bisection bandwidth for WDC12 is 640GB/s.

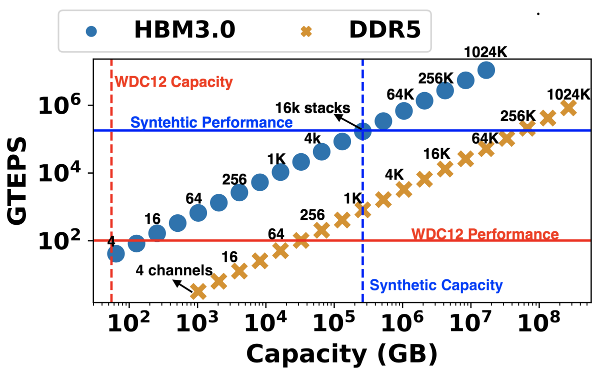
Answer: Using our model, we find that 8 accelerators each with one stack of HBM3.0 as vertex memory and one DDR5 channel as edge memory in a near memory processing configuration is the best for the given performance target (100 GTEPS) and the graph (WDC12).

3.2 Use Case 2: Locality Calculation

The proposed model can be used to calculate the locality requirement (i.e., α) for the accelerator while using homo-



(a) Edge memory. Desirable region: right-side of capacity limit.



(b) vertex memory. Desirable region: Upper right quadrant.

Fig. 2: Performance vs. capacity: (a) Edge memory the dotted vertical line indicates the required edge capacity. (b) Vertex memory Uses the performance (TEPS) in addition to the required vertex capacity (dotted horizontal line). We consider $\alpha = 1$.

geneous memory technology for edges and vertices. In this example we consider using only DDR5 devices.

From our previous calculation we will use eight accelerators, and we will now use two channels of DDR5 each (Figure 2b). Eight channels of DDR5 provides the required capacity for the vertex memory (right-side of dashed red line). Eight channels of DDR5 only provides 6.4 GTEPS with $\alpha = 1$. To increase the performance to 100 GTEPS our model determines α must be less than 0.064 using Table 1 Row 3. Thus, if the accelerator's on-chip memory for vertex information was a simple cache it would require a hit ratio of 93.6% for a balanced system with DDR5 channels for both edge and vertex with minimum over provisioning on the capacity. Thus, our model shows that in a homogeneous memory system, unless the accelerator can find significant locality, the vertex memory will be the bottleneck.

3.3 Use Case 3: System Evaluation

Next, we compare the results from our model to a real example of a highly-scalable graph analytics system. We use our model to estimate the performance of the new Sunway supercomputer running BFS on a synthetic graph with 17.56 trillion vertices and 281 trillion edges and compare our result with the performance reported by Cao et al. [6].

Sunway supercomputer is equipped with 40 million processing cores across 103,912 processing units each with eight DDR3 channels. Using our model we predict a performance of 71,680 GTEPS with the Sunway supercomputer without any optimization ($\gamma = 1$, $\alpha = 1$). However, Cao et al. achieves 180,792 GTEPS (2.5$\times$ better performance). This discrepancy is due to their novel 3-D partitioning method which leads to a lower γ ($\gamma \approx 0.4$).

Furthermore, we can use our model to create a *balanced* system for the same synthetic graph and a performance target of 180,792 GTEPS and compared our system against Sunway supercomputer. The proposed balanced system from our model requires 256 TiB of memory for storing the vertices and 2 PiB of memory for storing the edges. The model shows the balanced design only needs 8192 accelerators each with 2 DDR5 channels and 2 HBM3.0 stacks for edge and vertex memory, respectively. The system derived using our model uses 12$\times$ fewer processors (accelerators tiles) and 4$\times$ smaller memory by taking advantage of heterogeneous memories to better match the capacity-bandwidth tradeoffs inherent in graph processing.

4 Conclusion

As the size of the graphs increase, by orders of magnitude in the future, simulation is no longer viable for conducting a system-level design space exploration of graph accelerators. We need an analytical model that can help an architect make high level design decisions such as number of tiles, what is the target network bisection bandwidth, what mix of memory technologies make sense, what are the trade-offs between a disaggregated memory configuration and a near-memory configuration, and how to build a balanced system for a given graph size and given performance target. In this paper we present a high level performance model for large scale graph processing and how to use this model to answer these questions.

Currently the proposed model targets asynchronous vertex-based graph programming paradigms. In the future we plan to extend this model to linear algebra based formulation of graph analytics [12] and dynamic graphs and use this model to drive the design of the microarchitecture of the accelerator tile itself.

References

[1] "Giraph - Welcome To Apache Giraph!" [Online]. Available: https://giraph.apache.org/
[2] "ISS Group at the University of Texas." [Online]. Available: https://iss.oden.utexas.edu/?p=projects/galois
[3] "WDC - Hyperlink Graphs." [Online]. Available: http://webdatacommons.org/hyperlinkgraph/
[4] J. Ahn *et al.*, "A scalable processing-in-memory accelerator for parallel graph processing," in *ISCA*, 2015.
[5] S. Beamer *et al.*, "Gail: The graph algorithm iron law," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, 2015.
[6] H. Cao *et al.*, "Scaling graph traversal to 281 trillion edges with 40 million cores," in *PPoPP*, 2022, pp. 234–245.
[7] V. Dadu *et al.*, "Polygraph: exposing the value of flexibility for graph processing accelerators," in *ISCA*, 2021.
[8] T. J. Ham *et al.*, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *MICRO*, 2016.
[9] Y. Low *et al.*, "GraphLab: A New Framework For Parallel Machine Learning." [Online]. Available: http://select.cs.cmu.edu/code.
[10] A. Mukkara *et al.*, "Exploiting locality in graph analytics through hardware-Accelerated traversal scheduling," *MICRO*, 2018.
[11] S. Rahman *et al.*, "Graphpulse: An event-driven hardware accelerator for asynchronous graph processing," in *MICRO*, 2020.
[12] N. Sundaram *et al.*, "Graphmat: High performance graph analytics made productive," *arXiv preprint arXiv:1503.07241*, 2015.