Dynamically Balanced Request Latency of Memory Controller in NoC for GPGPU

Yongbin Gu, Yunfan Li, Ye Li, Wei Wang, and Sihe Yang
Oregon State University
School of Electrical Engineering and Computer Science
{guyo, liyunf, liuye, wangw5, yangsih}@oregonstate.edu

I. INTRODUCTION

In last decade, GPU architectural has experience four developments. Tesla architectural, Fermi architectural, Kepler architectural, and Maxwell architectural. From Fermi architectural and Kepler architectural to Maxwell architectural, the fabrication process and the thermal control system efficiency has been improved, and more CUDA core can be put in the GPU.

Before Direct9X, in the field of game image, the Transform and Lighting was controlled by CPU, and the performance is determined by the number of Transform and Lighting pipeline. There were two vital basic element in GPU architectural that is Vertex Shader and Pixel Shader. The two Shaders can only work independently. In DirectX10, Microsoft use Stream Processor instead of Vertex Shader and Pixel Shader. After Fermi architectural era, the Stream Processor is called Compute Unified Device Architecture (CUDA Core). CUDA Core and other components compose Streaming Multiprocessor (SM), which is the basic mod. In the SM, there are Special Function Unit, Instruction Cache, Warp Scheduler, etc. From Fermi to Maxwell GPU architectural, most of improvements were adjustment in SM, such as pile up CUDA Core and other components [1]. In Fermi architectural, only 32 CUDA Cores in SM. Kepler architectural has 192 CUDA Cores and there are 128 CUDA Cores per SM in Maxwell architectural. Polymorph Engine has been import in Fermi architectural era and only design for DirectX 11. PolyMorph includes Vertex Fetch, Tessellator, Viewport Transform, Attribute Setup and Stream Output. PolyMorph Engine can translate data to 3D module, then Raster Engine display 2D image in screen, which translate from 3D module.

In Kepler architectural, the most obvious change is the fabrication process from 40nm to 28nm. So in the same area, Kepler architectural can assemble more composes. More LD/ST units in SM than Fermi's architectural. Moreover, Shader frequency division technology can exchange large amount of throughput through less computing resources, but the shortcoming is that it can generate large amount of heat to prevent high frequency progress of GPU [10]. In Kepler architectural, NVIDA introduce GPU Boost function, which can auto adjust the GPU's working condition.

Maxwell architectural GPU did not change too much, but they rearranged SMM. SMM is the same with SM. The number of CUDA core in SMM decrease to 128 from 192(Kepler architectural), and the SMM has been cut into four regions. Every small regions have independent Control Logic. So every CUDA Core in Maxwell Core performance is 1.35 times more than Kepler architectural and Watt Performance is 2 times than Kepler architectural [8].

Even if GPU is mainly designed for graphics applications, it also has a potential performance to execute non-graphics [1, 9, 13] because of its teraflop barrier [2, 14] and other optimizations. According to four experimental simulations of CUDA, a GPU product of NVIDIA Corporation, it shows the following improvements. Firstly, GPU simulator (GPUPU-Sim) executes twelve common applications of CUDA to study the characteristics of CUDA. Then, compared with bisection bandwidth, the non-graphics applications have a tendency to be less sensitive to the latency. Thirdly, based on reducing threads which is simultaneously executed by specific applications, decreasing contention shows a way to improve performance. Fourthly, the article shows the analysis of the dynamic instruction mix, SIMD wrap branch divergence properties and DRAM locality characteristics. As a result, GPU simulator is beneficial to study GPU architecture in future.

Based on CUDA programming model [12, 13], it shows following structures. GPU can be regarded as a computing kernel of parallel structure. The kernel consists of a grid of scalar threads. The blocks are organized by a group of threads in a gird. These blocks also dubbed cooperative thread arrays (CTAs) [9]. Any CTA threads can concurrently get access to the shared memory. Thus, these structures comprise the overview of GPU simulation. For the baseline architecture, the shader cores, termed as streaming multiprocessor (SM) [13], parallelly handle the data to the interconnection network in GPU. With respect to this simulation, each shader core includes a SIMD with the width of 8 and a 24-stage. Meanwhile, the 24-stage pipeline needs more than 192 active threads to prevent from stalling. Assuming that each pipeline has 6 stages (fetch, decode, execute, memroy1, memory2, writeback), and each superpipeline has 4 degree, a warp includes a collection of 32 threads in the SIMD pipeline [9], these threads concurrently handle the same instruction over 4 successive clock. By judging branch divergence, executing branch is termed as "not taken", otherwise is termed as "taken" underlying the immediate post-dominator reconvergence mechanism [5]. Beside, threads should utilize memory space to run on the GPU simulation instead of memory regions (global, local, constant, texture, and shared [13]) on the real GPU.

GPU performance can be promoted with four proposals. Firstly, using a 4D blocking address diagram replaces low efficient linear space [7], it improves the spatial locality based on the 2D space. Secondly, loading repeatedly data could optimize DRAM efficiency, caused by the DRAM bandwidth bottleneck [16]. Then, rescheduling the threads which should not be wait for the end of long latency in the shader core and sending it to the SIMD pipeline is one of improvements. The last scheme is that substituting separate logical networks for the single interconnection network could prevent from protocol deadlock and accelerate memory request [3].

II. BACKGROUND

A. *Network On-Chips (NoCs)*

On-chip Networks (OCNs or NoCs) are one type of interconnection networks, which has high scalability, and have recently attracted considerable research attention. Combining together topological, routing algorithm and flow control design decisions for fast cycle times, area and power constraints important in NoC design field.

1) Topology

The first step in network design often is the topology design which determines the arrangement of the channels and nodes in the network. Furthermore, topology determines the number of hops and the complexity of implementation, which has significant impact on network cost-performance. The number of hops effect on network latency, throughput and energy consumption seriously. Moreover, switch degree that the number of links at a node, hop count that the number of hops a message takes from its source to destination, maximum channel loads the estimated max bandwidth the network can support and path diversity the multiple shortest paths between source and destination pair are the four main abstract metrics to evaluate performance and cost of topology. We have direct and indirect types of topologies that each router is associated with a terminal node and routers are distinct from terminal nodes. Tours, mesh, butterfly and hierarchical are the four common topologies up to date.

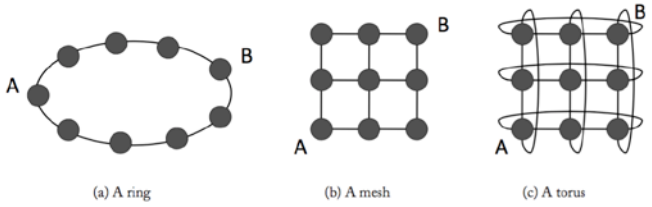

Figure 1. Common on-chip topologies: ring, mesh and tours.

In fig.1 (b)-(c) shows the direct topologies mesh and tours networks than can be described as k-ary n-cube. A mesh network is a torus network removed the end-around connections. With a torus network, all nodes have the same degree; however, with a mesh, the nodes at the edge of the network have a lower degree than nodes in the center of the network. Without the end-around connections of a tours, the average minimum hop count of a mesh is slightly higher. Either mesh and torus networks all provide rich path diversity to rout mess.

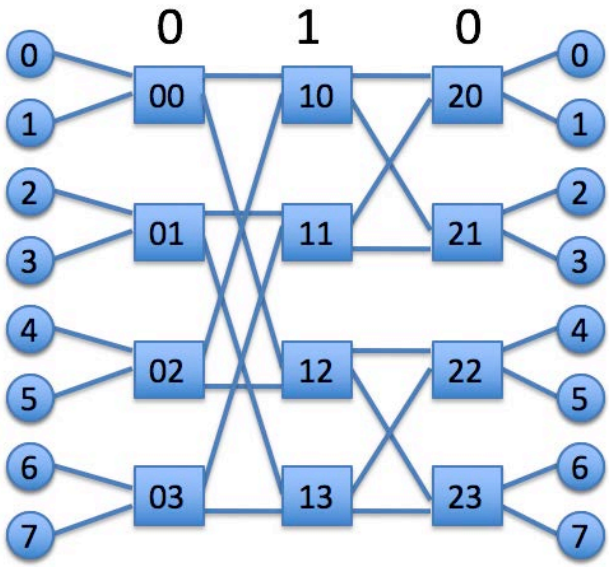

Figure 2. 2-ary 3-fly 2 input switch, 3 stages Butterfly Network.

A butterfly network shown in Figure2 is an example of indirect topology, can be described as 2 (k) -ary 3 (n) -flies that k is the degree of the switch and n is the number of stages of switches. Unlike the mesh or the torus network where the hop count varies based on source-destination pair, every source-destination pair in a butterfly network experiences the same hop count given by n – 1.

2) Routing

After fixing the topology that determines the connectivity of the network, the routing algorithm is used to determine the paths through network that a message will take through to reach its destination. The goal of routing algorithm is to distribute traffic evenly among paths provided by the topology network, and to avoid hot spots and contention, thus improving network throughput. Routing algorithms are generally divided into three classes: deterministic, oblivious and adaptive. Recently, the most commonly used routing algorithm in NoC is dimension-ordered routing(DOR) due to its simplicity. Furthermore, routing algorithms also have minimal and non-minimal classes. Minimal routing algorithm will choose the path with the minimal hops between its source and destination. Non-minimal routing algorithm has multiple paths chooses can be selected from its source and destination that probably will increase the number of hops, routing latency and also power consumption due to additional routers and links needed to be traversed by a message. Moreover, beside the consideration of delay, power consumption and throughput in designing a routing algorithm, guaranteed deadlock freedom is another key point in most applications. A deadlock can be occurred if there are no routing restrictions that a resource cycle can be produced.

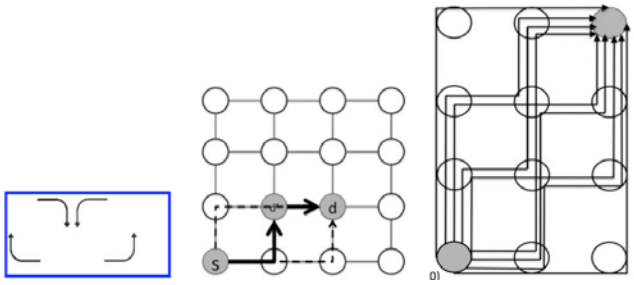

Figure 3. (a) one deterministic X-Y routing in 2D mesh network, (b) Oblivious Routing example, (c) Adaptive Routing example.

A routing algorithm can be described by which turns are permitted. In Figure 3 (a) shows the X-Y routing, one kind of deterministic Dimension-Ordered Routing (DOR) algorithm, can allow messages traveling north or south to turn east or west. DOR is obviously simple and deadlock-free. However, it eliminates paths diversity in a mesh network and decreases network throughput.

The oblivious routing decision is made without regard to network state, which is simple but unadaptable. In minimal oblivious routing, Figure3 b shows the intermediate destination d' is selected within the minimal quadrant formed by s and d, which has the minimum hop counts. Finally, Valiant's and Minima Oblivious routing algorithms are deadlock free when used in conjunction with X-Y routing; however, oblivious are not deadlock free when randomly choose between X-Y and Y-X turns.

Adaptive routing uses network state the local or global information in the local router to make routing decisions. Figure 3.b shows all possible paths that a message can take from its source node to the destination node. There are nine possible paths. An adaptive routing algorithm that leverages only minimal paths could exploit a large degree of path diversity to provide load balancing and fault tolerance.

Figure 4 shows the microarchitecture of the three-stage main network router. During the first pipeline stage, the incoming flit is buffered (BW), and in parallel arbitrates with the other virtual channels (VCs) at that input port for access to the crossbar's input port (SA-I). In the second stage, the winners of SA-I from each input port arbitrate for the crossbar's output ports (SA-O), and in parallel select a VC from a queue of free VCs (VS). In the final stage, the winners of SA-O traverse the crossbar (ST). Next, the flits traverse the link to the adjacent router in the following cycle.

3) Flow Control

After determining the topology and routing algorithm, the flow control determines allocation of network buffers and links to messages as they traverse network, which significantly impacts on throughput and latency of network. When a message is injected into the network, it's segmented into packets firstly, then packets are segmented into flits that short enough for flow control units. Different flow control

techniques based on different granularity: message-based, packet-based and flit-based.

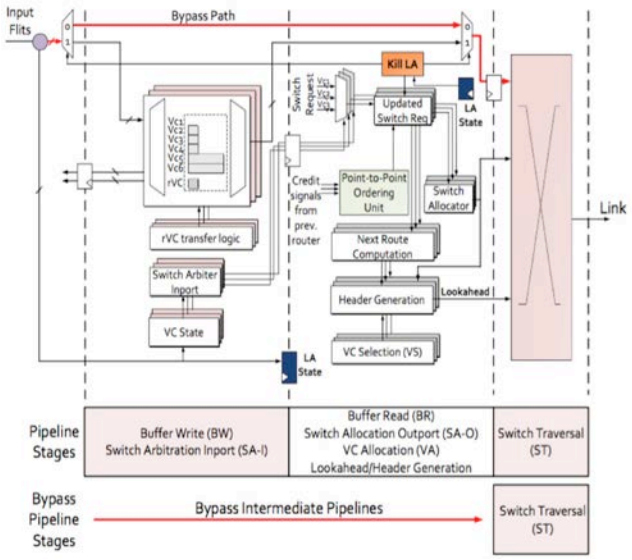

Figure 4. Router Microarchitecture

B. Memory Controllers (MCs)

Memory Controller (MC) is a kind of digital circuits that manage the data flow of the memory. Based on the different requirements, the memory controller can be an independent chip or integrated into other chips. In the GPGPU, the memory controller is integrated into the GPU chip, which has the potential to increase the chip performance and to decrease the system cost. The memory controller usually consist of the logic that is necessary to read and write the DRAM. And the "refresh" function is also included, because DRAM will lose data held in it without constant refresh. With correctly selecting the bank, row, column of the data address in the memory, the MC can read and write data from the memory. Memory controller bus widths range from 8 bit to 512 bits. In GPGPU, especially in the Fermi architecture of Nvidia, there are 8 memory controllers in a chip that can operate in parallel. This design can meet the requirement of the frequent reading and writing of the memory in modern GPU. Computer memory controllers have developed into a number of categories. These are Double data rate memory, Dual channel memory, fully buffered memory and flash memory controllers.

III. MOTIVATION

Since Graphics Processing Units (GPUs) have gotten a significant improvement in performance, it is widely used as co-processors of Central Processing Units (CPUs) not only in traditional desktop system, but also in High-Performance Computing (HPC) field. There are four major parts in GPUs which are named shader core system, cache system, DRAM system and On-Chip Network (a.k.a NoC) respectively. To enable the growing number of threads concurrently running in GPUs, it is imperatively to design a NoC that can support a large amount of on-chip communication cost-effectively. There are two exactly same physical networks exist in GPU, one of two carry memory or L2 cache access traffic travelling from shader cores to memory controllers which called request

network, the other network carry memory or L2 cache data information which travel back from memory controllers to shader cores, corresponding to request network, this is called reply network. Request data usually just contains 20 to 25 bytes memory address information but reply data at least includes 128 bytes (same size as cache line) which originally stored in main memory or L2 cache. So reply traffic is nearly 5 times of request traffic, which intuitively means that reply network is much more crowded than request network. But actually, Figure 1 shows that request network latency is much higher than reply network latency. This is because a cache line usually needs consume 9 cycles (under 16 bytes network bandwidth) to be fully injected into reply network, which means that memory controllers can only send one cache line every 9 cycles, but request packets just need to take 1 or 2 cycles to pop from request network into memory controllers, so memory controllers will eventually cannot accept request data anymore and thus request data will be stalled in request network worsening, we call this situation “backpressure”. In order to solve backpressure problem, reply network must be accelerated so that memory controller can send reply cache line more than current design and thus consuming more request data to remit request network traffic pressure and reduce overall network latency which packets travel in NoC

REFERENCES

[1] Advanced Micro Devices, Inc. ATI CTM Guide, 1.01 edition, 2006.

[2] Advanced Micro Devices, Inc. Press Release: AMD Delivers Enthusiast Performance Leadership with the Introduction of the ATI Radeon HD 3870 X2, 28 January 2008.

[3] Bakhoda, Ali, George L. Yuan, Wilson WL Fung, Henry Wong, and Tor M. Aamodt. "Analyzing CUDA workloads using a detailed GPU simulator." In Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on, pp. 163-174. IEEE, 2009.

[4] Duato, Jose, Sudhakar Yalamanchili, and Lionel M. Ni. Interconnection networks: An engineering approach. Morgan Kaufmann, 2003.

[5] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt. Dynamic warp formation and scheduling for efficient GPU control flow. In Proc. 40th IEEE/ACM Int’l Symp. on Microarchitecture, 2007.

[6] Jerger, Natalie Enright, and Li-Shiuan Peh. "On-chip networks." Synthesis Lectures on Computer Architecture 4, no. 1 (2009):1-141.

[7] Z. S. Hakura and A. Gupta. The design and analysis of a cache architecture for texture mapping. In Proc. 24th Int’l Symp. on Computer Architecture, pages 108–120, 1997.

[8] Harris, Mark. "5 Things You Should Know About the New Maxwell GPU Architecture." Parallel Forall. 21 Feb. 2014. Web. 16 Feb. 2016.

[9] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro, 28(2):39–55, 2008.

[10] McClanahan, Chris. "History and evolution of gpu architecture." *A Survey Paper* (2010).

[11] Morgan, Timothy Prickett. "Nvidia's Kepler Pushes Parallelism up to Eleven." • The Register. 15 May 2012. Web. 16 Feb. 2016

[12] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable Parallel Programming with CUDA. ACM Queue, 6(2):40–53, Mar.-Apr. 2008.

[13] NVIDIA Corporation. NVIDIA CUDA Programming Guide, 1.1 edition, 2007.

[14] NVIDIA Corporation. Press Release: NVIDIA Tesla GPU Computing Processor Ushers In the Era of Personal Supercomputing, 20 June 2007.

[15] Patterson, David A., and John L. Hennessy. "Computer organization and design." MorganKaufmann (2007):474-476.

[16] S. Ryoo, C. Rodrigues, S. Stone, S. Baghsorkhi, S.-Z. Ueng, J. Stratton, and W. W. Hwu. Program optimization space pruning for a multithreaded GPU. In Proc. 6th Int’l Symp. on Code Generation and Optimization (CGO), pages 195–204, April 2008.