A Memory-Efficient NoC System for OpenCL Many-core Platform

Chien-Hsuan Yen, Chung-Ho Chen, and Kuan-Chung Chen
Inst. of Computer & Communication Engineering, National Cheng Kung University, Tainan, Taiwan, R.O.C.
impromptu72@gmail.com, chchen@mail.ncku.edu.tw, edi@casmail.ee.ncku.edu.tw

***Abstract*— We present a memory-efficient NoC system for OpenCL many-core platforms. By offloading the OpenCL kernel programs into the many-core processor, we reveal that the memory contention overheads have dramatically increased with the scaling of the system, resulting to the poor performance scalability of the many-core system. We explore a memory-efficient NoC system design which includes a mesh network, a hybrid network interface for packet composition and decomposition, and a memory controller with access scheduling capability. Our experimental results show that a simple memory access scheduling and caching approach can easily boost the performance of the NoC and memory system up to 20 percent by eliminating the memory controller contention problem.**

Keywords—DRAM access scheduling, full system simulation, many-core system, Mesh NoC, OpenCL.

I. INTRODUCTION

Due to the power consumption and heat dissipation problem coming with higher clock speeds, modern processor architectures take parallelism as an important way to increase the performance. In addition to the flourishing homogeneous computing technology, system integrators also want to take full advantage of the other computing resource in the computer systems, such as GPUs, DSPs, or other processors. OpenCL (Open Computing Language) is a programing language and API framework for programmers to achieve parallel computing on the heterogeneous compute devices [1].

However, the interconnection and memory performance plays an important role in the contemporary parallel computing system. As more and more cores are incorporated into a single chip, the packet-switched Network-on-Chip (NoC) has emerged as a convincing replacement of traditional bus-based on-chip interconnections [8]. A well-designed network interconnection makes efficient use of the scarce communication resources, providing high-bandwidth, low-latency communication between masters and slaves [6, 7].

Nevertheless, for a many-core system that executes parallel programs, the memory is also a crucial resource since the number of memory controllers used is typically constrained by the chip's I/O. Also despite the fact that the physical bandwidths provided by the interconnection and memory controllers are much higher than the average bandwidth requirement of the applications, the memory contention overheads, caused by using the memory controllers concurrently, may increase with the scaled system dramatically and lead to the poor scalability of the many-core system. We demonstrate that even a simple DRAM access scheduling strategy is a promising solution to cope with the memory contention problem for an OpenCL many-core platform.

In this paper, we design a configurable mesh NOC system to explore the scalability issues of a many-core system. We evaluate the performance of the proposed NoC many-core system on the OpenCL supported full system simulation platform [3]. To alleviate the memory contention problem, we integrate the DRAM memory access scheduling policy with the proposed NoC system to enhance the memory efficiency and scalability.

The rest of this paper consists of the following sections. In Section II, we briefly introduce the OpenCL framework and the full system many-core platform with our NoC system. Section III discusses the memory access scheduling scheme in detail and its enhancements. Section IV presents the evaluation system and results. We conclude this paper in Section V.

II. OPENCL PLATFORM AND NOC SYSTEM OVERVIEW

In this section, we first introduce the virtual many-core platform that executes the offloaded OpenCL programs and followed by the proposed NoC system.

A. OpenCL Framework and Many-core platform Overview

Fig. 1 shows the architecture of the full system many-core platform. In our OpenCL system framework, the virtual compute device is an ESL many-core platform and the compute unit is an ARMv7a instruction set simulators (ISS), caches also included. Hence, the OpenCL runtime offloads a work-group to one of the ARM cores in the many-core system, where the interconnection for these compute units is our NoC system.

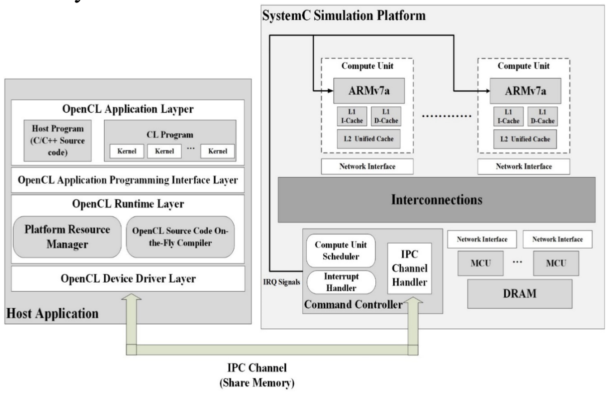

Fig. 1. OpenCL System Simulation Platfrom

B. OpenCL Memory Management

The memory hierarchy defined by the OpenCL includes _global, _constant, _local, and _private memories. The global memory and the constant memory can be used by all the compute units of a compute device. In contrast, a local memory only belongs to a compute unit. Then, the host processor communicates with these compute devices via sharing the host main memory to the compute device memory. To simulate the OpenCL memory hierarchy, the Memory Management Unit (MMU) transforms the virtual address of the private local memory pages of a work-group into to the unique private physical memory of the active ARM core. Since each core exclusively accesses their local memory region, we arrange these local physical memory into an interleaving memory bank according to the core ID, as shown in Fig. 2 where four memory controllers (MCU) are shown. The Global memory and the Instruction memory are also allocated to the different memory space.

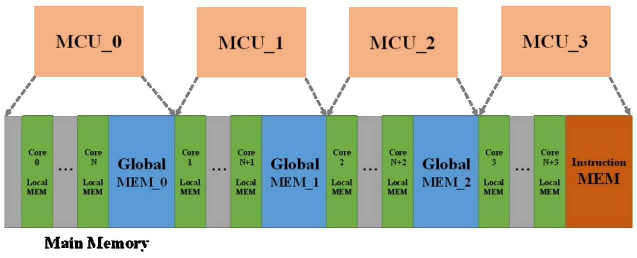

Fig. 2. Memory Management and Allocation

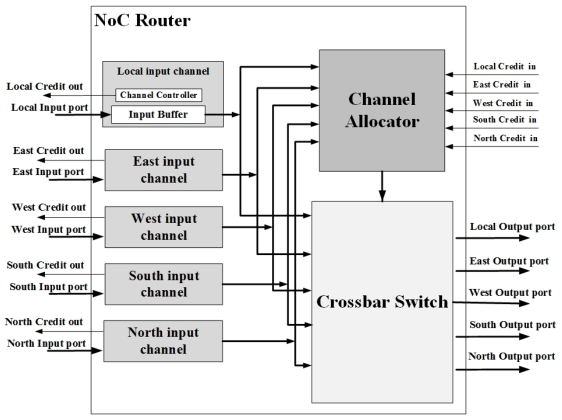

Fig. 3. Router Architecture

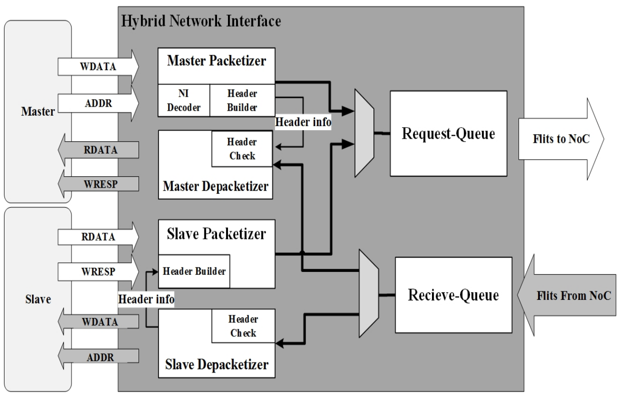

Fig. 4. Architecture of the Hybrid Network Interface

C. *Network-on-Chip System*

In the Mesh NoC, each node in the network is a router, which connects with the neighboring routers with four duplex channels. A node has a local channel connecting to either an ARM core or a memory controller through a network interface [9]. In the NoC system, a complete access to a slave consists of a request packet and a response packet. A packet consists of a 32-bit header flit and more than one 32-bit payload flits. The traditional wormhole flow control uses extra flags in each flit to indicate the flit type, which may be the header, the payload, or the tail [5]. Our design has removed these redundancy associated with each flit, including the tail flit. To do this, the router logic in each mesh node has to acquire the information of the packet length from the operation type field and burst length field from the header. To recognize a whole packet in the flit stream, the channel allocator, in the router, only has to count the number of the forwarded flits instead of identifying the tail flit. In this way, our design has achieved the wormhole flow control without adding a tail flit and flit headers.

In the following, we give the detailed description of each component in the NoC system:

1) Router Achitecture : A router consists of five input channels, five output ports, a channel allocator, and a crossbar switching fabric as depicted in Fig. 3. The crossbar switch fabric passes the packets to the five output ports flit by flit according to the decision of the channel allocator logic. The channel controller in each input channel delivers a credit signal to the channel allocator of the connected router to inform whether the buffer is full or not. Then, the channel allocator reads the header flits from the five input channels and receives the five credit signals from the neighboring routers. At the beginning, the channel allocator gets the information from each header, including request type, burst length, and target location. Next, it selects one output port for the received packet according to the routing algorithm. Then, it arbitrates the use of the output port that has more than one output packet and then multiplexes the crossbar switch.

2) Hybrid Network Interface : The network interface illustrated in Fig. 4 transforms the request command from a master into a request packet or reads the received packet for the local node. The hybrid network interface (HNI) consists of a request-queue, a receive-queue, two packetizer logics, and two depacketizer logics. The HNI can serve both master and slave concurrently with the same design by sharing the request-queue and receive-queue. As the HNI receives the request from the master, the master packetizer transforms the request control signal into a header flit, and follows by the payloads. In the master packetizer, there is a mapping table to identify the location for all of the slaves. Thus, the NI Decoder looks up the mapping table to acquire the slave location for the request address. Finally, the request-queue will pass the packets to the mesh network flit by flit. When a request packet reaches the targeted slave, the depacketizer interprets the packet for the slave. Furthermore, the header check unit in the depacketizer logics will check whether the packet is misrouted. Finally, the depacketizer delivers the data to the master if it is a correct packet.

III. Memory Access Scheduling Scheme

Modern DRAM has a 3-D structure, bank, row, and column selections. To maximize the memory bandwidth of DRAM, the modern DRAM architecture allows to pipe the memory operations [2]. It can process different memory operations in parallel on different banks independently, and there is a row buffer in each memory bank to cache the recently used row data. This method makes the best use of the peak memory bandwidth, but also leads to a close relationship between the memory access order and the DRAM performance.

The competition among the processors to access a memory controller may be serious since it is a many-to-few access system. We take advantage of the packet-switching characteristic of the NoC to implement a straightforward DRAM access scheduling mechanism in the memory controller. To utilize the memory bandwidth more efficiently, our memory scheduler gets improvement from the following two aspects: 1) reducing the row buffer miss rate in DRAM by reordering the memory requests, 2) reducing the number of DRAM accesses by caching the latest cache line read by the memory controller.

A. *Priority-Aging Hit-First Memory Access Scheduler*

We employ a "Hit-First" DRAM access scheduling policy, which arbitrates all the DRAM requests according to the request address[4]. There is a register in the memory scheduler to record the address region of the current row buffer in the DRAM bank. If the DRAM request in the queue buffer hits the current row buffer, the memory scheduler will give this request higher priority, otherwise it just takes the FCFS policy.

However, only the Hit-First policy may lead to starvation for the queued requests if the latter continually hits on the same DRAM row buffer. In this way, the queuing delay of the early reached requests, which access to the different row buffer, will be too long and the parallel program may be stalled by this unbalanced DRAM service. To address this problem, we integrate a priority-aging policy to the original Hit-First scheduler to eliminate the possibility of starvation. If the DRAM scheduler finds one of the aging-counter is over a programmable limited-age, which is set to 64 for not affecting the 90% of the DRAM commands for the OpenCL applications, it will give the highest priority to this command no matter it is a row miss or not.

B. *Last-Read Buffer*

We also use a last-read buffer to cache the last read data. If other cores have to read the same cache line, the memory controller just forwards the cached cache line instead of requesting the DRAM again. This last-read buffer reduces the redundant accesses currently in the command buffer.

IV. Evaluation Results

This section explores the performance impact of the interconnection and the memory system.

A. *System Configuration*

We evaluate the OpenCL full system simulation platform with seven OpenCL benchmarks, including five NVIDIA applications and two AMD applications. We choose these applications as our benchmarks for the reason that they have more work groups in a kernel command, utilizing all the ARM cores in the many-core system more effectively.

Table 1 shows the system configuration of the full system simulation platform and the memory-efficient NoC system. For the network arrangement of cores and memory controllers, we distribute the four memory controllers toward the corners of the mesh and place the cores symmetrically among the network to alleviate the traffic contention in the center of the mesh network.

TABLE I. System Configuration

Virtual Platform Configuration	
Platform Component	**Configuration**
No. of Cores	1, 4, 8, 16, 32, 64
Processor Model	ARMv7a ISS with VFP 3.0@ 1.25GHz
L1 I/D-Cache	2-Way, 64B line, Size 32KB, 3-cycle delay
L2 Unified Cache	8-Way, 64B line, Size 256KB, 20-cycle delay
DRAM Model	512MB@166MHz x4, TCL-TRCD-TRP all 10 ns
Interconnection	Configurable Mesh NoC
Memory-Efficient NoC System	
Topology	Mesh, 3x3, 5x5 7x7, 9x9 nodes
Flow Control	Wormhole
Routing Algorithm	XY Routing
Flit Width	32-bit
Input Channel Depth	20 flits
Channel Allocator Performance	1.25GHz
Channel Bandwidth	80 Gbps (full duplex)
DRAM Access Schedule Policy	Priority-Aging Hit-First

Fig. 5 illustrates the memory access flow and the definition of the timing parameters. The Memory Access Time (tMAT) is the total latency to acquire data from the memory system for a processor executing a benchmark. The NoC Latency (tNL) represents the total interconnection latency that consists of the NoC transmission latency and the queueing delay in the command buffer of the target memory controller. The DRAM Latency (tDL) is the total DRAM operation time for a particular core.

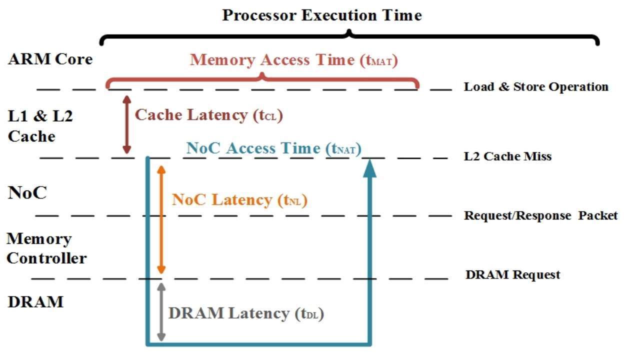

Fig. 5. Memory Access Flow Diagram in the NoC system

B. *Performance Evaluation*

Fig. 6 shows the breakdown of the memory access time for 64-core system in terms of the cache latency, the NoC latency, and the DRAM latency for the baseline (native) system which employs no optimizations. All the OpenCL applications take over 75% of the memory access time on the NoC latency. Despite the fact that we have arranged the private memory of all the processors fairly to the four MCUs, the congestion for the global memory is still very intense.

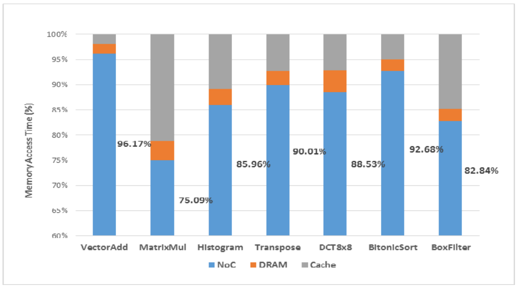

Fig. 6. Breakdown of the Memory Access Time for 64-core System

In order to enhance the memory performance, we add the "Priority-aging Hit-first" memory scheduler and last-read buffer into the NoC system. Fig. 7 shows the total DRAM latency reduction compared with the native NoC in 64-core system. The improvement from the Hit-First policy is about 3% to 9% while the benefit from the last-read buffer is more obvious, up to 15%. Whenever the Hit-First policy prevents a request from the redundant overhead of T_{RP} and T_{RCD}, it can save about 20ns to 40ns. Instead, when the last-read buffer reduces a redundant DRAM request, it can save about 90ns to 120ns. Besides, the memory access scheduling approach can not only enhance the memory performance but also decrease the traffic congestion on the NoC system. Fig. 8 shows the total reduction of the NoC Access Time (t_{NAT}) normalized with the native NoC. As the number of the cores grows, the memory scheduler gains more improvement from the scaled system. In summary, the memory scheduler and the NoC system mutually benefit each other to enhance the overall performance for the many-core system.

V. Conclusion

We have proposed a memory-efficient NoC system to improve the scalability and memory performance for the many-core platform under the execution of the memory-sensitive OpenCL applications. By the breakdown of the memory utilization in the OpenCL programing model, we find that only providing a high bandwidth interconnection cannot prevent the tremendous memory contentions for a scaled system. As revealed, a simple DRAM access scheduling approach coupling with the NoC system can effectively enhance the memory performance and utilization efficiency. The scalability for a many-core system under execution of the memory-intensive applications can be achieved by the cooperation of the packet-switched NoC system and a simple DRAM access scheduling scheme with a last-read cache buffer.

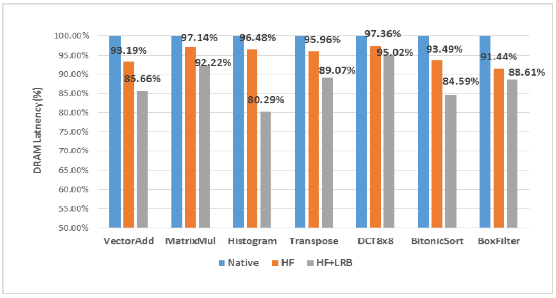

Fig. 7. Reduction of the DRAM Latency Normalized with the Native NoC in 64-core System

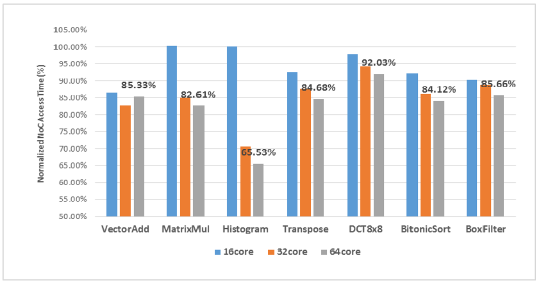

Fig. 8. Reduction of the NoC Access Time

VI. Acknowledgments

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 103-2221-E-006 -226 -MY3.

References

[1] Khronos OpenCL Working Group, "The OpenCL Specification Version 1.2," 2012, http://khronos.org/opencl.

[2] S. Rixner, W.J. Dally, et al., "Memory access scheduling", in *Proc. of the 27th International Symposium on Computer Architecture,* ISCA, 2000, pp. 128-138.

[3] K.-C. Chen and C.-H. Chen, "An OpenCL Runtime System for a Heterogeneous Many-Core Virtual Platform," in *IEEE International Symp. on Circuit and Systems* (*ISCAS'14)*, Jun., 2014, pp. 2197-2200.

[4] H.-Z. Zheng, J. Lin, et al., "Memory Access Scheduling Schemes for Systems with Multi-Core Processors," in *Proc. of the 37th International Conference on Parallel Processing*, *ICPP'08*, 2008, pp. 406-413.

[5] F.-A. Samman, T. Hollstein, and M. Glesner, "Networks-on-chip based on dynamic wormhole packet identity mapping management," ACM VLSI Design, No. 2, Jan. 2009.

[6] W.J. Dally, and B. Towles," Principles and Practices of Interconnection Networks," Morgan Kaufmann Publishers, 2004.

[7] P.P. Pande, C. Grecu, et al., " Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures," *IEEE Transactions on Computers*, Vol. 54, No. 8, Aug., 2005, pp. 1025-1040.

[8] D. Wentzlaff, P. Griffin, et al., "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Computer Society*, Vol.27, pp. 15-31, 2007.

[9] M. Daneshtalab, M. Ebrahimi, and P. Liljeberg, "Memory-Efficient On-Chip Network with Adaptive Interfaces," in the *IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems (TCAD)*, Vol. 31, No. 8, pp. 146-159, Jan. 2012.