2011 International Conference on Parallel Architectures and Compilation Techniques

An Alternative Memory Access Scheduling in Manycore Accelerators

Yonggon Kim, Hyunseok Lee, John Kim
Department of Computer Science, KAIST
{*ilios,leehseok,jjk12*}*@kaist.ac.kr*

***Abstract*—Memory controllers in graphics processing units (GPU) often employ out-of-order scheduling to maximize row access locality. However, this requires complex logic to enable out-of-order scheduling compared with in-order scheduling. To provide a low-cost and low-complexity memory scheduling, we propose an alternative memory scheduling where the memory scheduling is performed not at the destination (i.e., memory controller) but is done at the source (i.e., the cores). We propose two complementary techniques in source-based memory scheduling – network congestion-aware source throttling and *superpackets*, where multiple request packets are grouped together to create a single superpacket. By combing these techniques, the performance across a wide range of application is within 95% of the complex FR-FCFS on average and at significantly lower cost and complexity.**

I. Introduction

In many-core accelerator architectures such as graphics processing units (GPU), the memory system is a shared resource among the large number of cores and threads. As the number of cores increases, the memory system shared by the cores/threads not only includes the memory controllers but also the on-chip network that need to be traversed to reach the memory controllers. Within the memory system, memory scheduling, which is performed in the memory controllers, has significant impact on overall system performance. To increase memory scheduling efficiency, an out-of-order scheduler or First-Ready First-Come First-Serve (FR-FCFS) [2] is commonly used but FR-FCFS requires a complex structure as a fully-associative comparisons are required. In this work, we propose an alternative memory scheduling to reduce the complexity of the memory scheduler by making the memory scheduling decision at the *source* or the cores, instead of conventional memory scheduling done at the destination or the memory controllers.

The source-based memory scheduling exploits two techniques: congestion-aware source throttling and superpackets. Since the high on-chip network traffic caused by large memory access limits overall performance, we propose source throttling through *congestion-aware* memory scheduling and describe a distributed congestion-aware memory scheduling which throttles memory requests locally at each core. We also exploit the observation from prior work [3] which showed that for non-graphics, highly-parallel (GPGPU) applications on manycore accelerators, high DRAM row buffer locality access pattern is observed at each shader core but the locality is destroyed in the on-chip network as packets become interleaved. Instead of modifying the on-chip network which increases the storage requirement and can adversely impact the router critical path [3], we propose to create *superpackets* at the source such that the row locality is maintained when the DRAM requests arrive at the memory controller.

II. Congestion-Aware Memory Scheduling

The memory-intensive applications evaluated in this work create high memory traffic and congestion at the memory controllers. This congestion creates network congestion – which in turn, impacts overall performance. To reduce network congestion, we propose source-throttling of memory requests to decrease the number of in-flight memory requests and reduce network congestion. The reduced network congestion translates into lower overall memory access latency while providing higher overall performance and also enables the complexity of the memory scheduler and on-chip network to be reduced.

We introduce a *distributed* congestion-aware algorithm (local congestion-aware (LCA) algorithm) as described below (Algorithm 1). Each core (c) maintains the number of outstanding requests ($r_c(m, b)$) injected by the local core for each MC (m) and each bank (b) and throttle based on this value. In LCA, since the shader cores operate at a warp granularity we add additional constraint that all requests within a warp must be *eligible* – i.e., each request meets the constraints of the LCA algorithm.

Algorithm 1 LCA algorithm

```
At each shader core (c)
if all req within a warp satisfy r_c(m, b) = 0 then
    for all req within a warp do
        inject req;
        r_c(m, b)++;
    end for
else
    throttle;
end if
```

In addition to LCA, we propose a modified LCA (mLCA) that is based on the LCA algorithm but further restrict or throttle the injection of packets into the network. We add a *core* constraint to LCA – if the total number of outstanding requests in the core exceed a threshold t, the core is throttled until replies are received.

1089-795X/11 $26.00 © 2011 IEEE
DOI 10.1109/PACT.2011.37

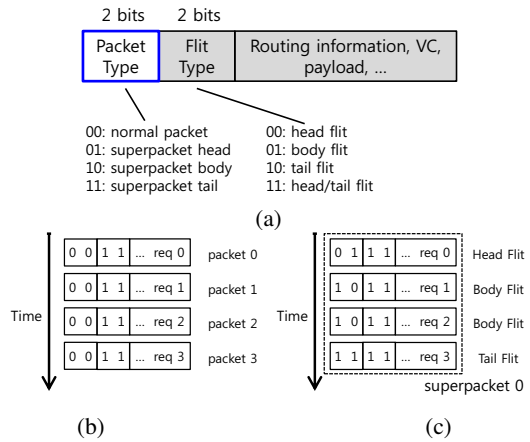

Figure 1: (a) Packet header change required to support superpacket with the highlighted bit showing the only change required. Example of injecting four requests using (b) normal packets and (c) a superpacket is also shown.

III. Superpackets

To avoid the loss of inherent row locality, multiple requests (and thus, multiple packets) injected from a single shader core can be grouped together to form a *superpacket* such that other requests are not interleaved when these requests arrive at the destination memory controller. The use of superpacket minimizes the need for a complex out-of-order memory scheduler to extract row locality at the destination. The only modification needed to support superpacket in an on-chip network is additional bits in the packet header to differentiate between a normal packet and a superpacket as shown in Figure 1(a). From the network perspective, the superpacket is identical to a *normal* packet but only with larger number of flits. At the ejection port of the destination router, however, each packet within a superpacket is partitioned into separate packets and transmitted to the memory controller.

An example of superpacket is shown in Figure 1(b,c). Baseline *normal* packets are shown in Figure 1(b), which consist of 4 requests that are injected as 4 separate packets into the network. These stream of requests can be interleaved with requests from other cores while traversing the on-chip network, even if they have row locality. However, with a superpacket formed as shown in Figure 1(c), all four requests are grouped together in a single packet and thus, regardless of the on-chip network arbitration, other requests cannot be interleaved since packet is the unit of routing in on-chip networks.

A superpacket can be formed based on the order of the requests that are considered at the core (in-order (I) or out-of-order (O)), source critieria (whether the requests only within a warp are considered (W) or all requests from the core are considered C), or destination critiera (MC-match (M) or row-match (R)) (Table I).

Config	Description
ICM	group all consecutive requests from a shader core to a given MC
ICR	group all consecutive requests to the same row
IWM	similar to ICM but only requests from the same warp can be grouped together
IWR	similar to ICR but only requests from the same warp can be grouped together
OCM	maximize the size of superpacket by maximizing the search space for the superpacket
OCR	maximize DRAM row locality from a shader core
OWM	N/A
OWR	N/A

Table I: Different configuration of superpacket.

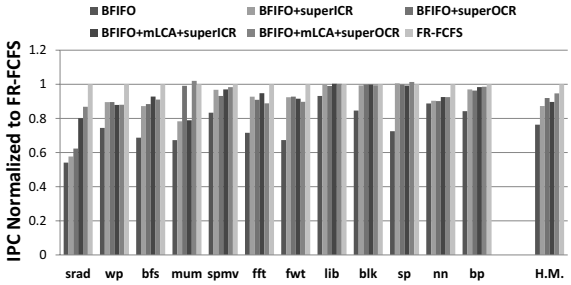

Figure 2: Performance of representative superpacket configurations and evaluation result of combining superpacket and congestion-aware memory scheduling.

IV. Preliminary Results

To evaluate our alternative memory scheduling, we use a detailed, cycle-level simulator (GPGPU-Sim) [1]. We use a 6×6 2D mesh network with 28 shader cores and 8 MCs. We use applications from NVIDIA's CUDA software development kit (SDK), benchmarks from Rodinia, Parboil, and from the set used by [1]. With superpacket, the ICR increases performance by 14% on average while OCR increase performance by 20% over BFIFO and achieves 92% of FR-FCFS, without requiring a complex out-of-order scheduler (Figure 2). The performance results of combining congestion-aware scheduling and superpacket are also shown in Figure 2. BFIFO with mLCA and OCR superpacket configuration is able to achieve 95% of the FR-FCFS results.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0015039).

References

[1] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *ISPASS 2009*

[2] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *ISCA 2000*

[3] G. L. Yuan, A. Bakhoda, and T. M. Aamodt. Complexity effective memory access scheduling for many-core accelerator architectures. In *MICRO 2009*