DRAM Scheduling Policy for GPGPU Architectures Based on a Potential Function

Nagesh B. Lakshminarayana, Jaekyu Lee, Hyesoon Kim, Jinwoo Shin
School of Computer Science, Georgia Institute of Technology
{nageshbl, jaekyu.lee, hyesoon.kim, jshin72}@cc.gatech.edu

Abstract—**GPGPU architectures (applications) have several different characteristics compared to traditional CPU architectures (applications): highly multithreaded architectures and SIMD-execution behavior are the two important characteristics of GPGPU computing. In this paper, we propose a potential function that models the DRAM behavior in GPGPU architectures and a DRAM scheduling policy, α-SJF policy to minimize the potential function. The scheduling policy essentially chooses between SJF and FR-FCFS at run-time based on the number of requests from each thread and whether the thread has a row buffer hit.**

Index Terms—**GPGPU, DRAM scheduling, Potential function**

I. Introduction

GPGPU computing architectures employ multithreading to hide memory access latency. Nonetheless, DRAM scheduling is an important component of such architectures for achieving high-performance for memory intensive applications and applications that have few threads. Hence, we propose a new DRAM scheduling policy, α-SJF, which considers two important GPGPU characteristics.

The first characteristic is the presence of large-scale multithreading. When a warp[1] blocks waiting for a memory access to complete, the (SIMD) core switches to a ready warp. As the number of ready warps varies at run-time, different cores can tolerate memory access latencies to different degrees. We quantify the ability to tolerate memory access latency by a metric called tolerance which is similar to slack or deadline. The proposed α-SJF uses tolerance for memory request scheduling by giving cores with low tolerance priority over cores with high tolerance.

The second characteristic is SIMD execution, because of which a single memory instruction may generate multiple memory requests. A warp could also execute multiple memory instructions before blocking. For a warp blocked on memory accesses, all its memory requests (may) have to be serviced for it to resume execution, otherwise, the warp may continue to be blocked or it may block again immediately due to dependencies on unserviced memory requests. α-SJF tries to schedule all requests from a warp as one group, so that the warp becomes ready in a short time. To achieve this objective, α-SJF employs the well-known Shortest-Job-First (SJF) scheduling policy. α-SJF represents each warp by a queue that holds all memory requests from the warp. These queues are considered as jobs whose durations are equal to the length of the queues.

However, unlike the FR-FCFS (First-Ready First-Come-First-Served) [5], [6], [8] policy, SJF does not exploit row buffer locality in memory requests. Thus, SJF can reduce the throughput of DRAM which could result in performance degradation for applications limited by the performance of DRAM. To consider all three aspects - tolerance, SIMD-execution, and row buffer locality - we propose an adaptive policy, α-SJF, where the parameter α explores the tradeoff in performance between SJF and FR-FCFS. This policy is based on a potential function developed by us for modeling GPGPU memory systems. The potential function is similar to α-MW (maximum weight) [7] algorithm in queuing systems.

[1]In NVIDIA architectures, 32 threads are executed together in lock-step as a warp and in AMD GPU architectures, 64 threads are executed together as a wave-front.

II. Potential Function

We propose a scheduling algorithm using the "Lyapunov-based scheduling strategy", i.e., we first define an appropriate Lyapunov-Foster potential function, and second, design a scheduling policy to minimize the potential in a dynamic system. At time t, the scheduler schedules the memory request that would minimize the potential value at time $t+1$. Here, the potential is essentially modeled as the time required to service all the requests in the system.

Posted to the IEEE & CSDL on 11/22/2011
DOI 10.1109/L-CA.2011.32

A. Potential Function of a Memory System

We first develop a potential function ($r(t)$) without considering the GPGPU characteristics explained earlier. For simplicity, we assume that there are only two cores, p and q, and that there is only one DRAM bank. The developed potential function can be easily extended to a system with more than two cores and multiple banks. In a multi-bank memory system, each bank schedules requests based on the potential of requests to that bank only.

The potential function uses the following notation:

- Every warp from each core has a queue.
- $q_i(t)$ (or $p_i(t)$): the number of memory requests in queue i of core q (or p) at time t.

The potential function without considering tolerance and SIMD-execution behavior is as follows:

$$r(t) = \sum_{i=1}^{N} q_i(t) + \sum_{i=1}^{N} p_i(t) \tag{1}$$

Therefore, if $r(t) \ > 1$

$r(t+1) = r(t) - 1$ if the serviced memory request hits in the DRAM row buffer.

$r(t+1) = r(t) - 1/m$ if the serviced memory request misses in the DRAM row buffer.

where, $m = \frac{\text{(service time of a request with DRAM Row buffer miss)}}{\text{(service time of a request with DRAM Row buffer hit)}}$

m denotes the penalty for serving a request with row buffer miss. Serving a row buffer hit takes the same time as a DRAM column access. While the time to serve a row buffer miss is the sum of precharge, row-access and column-access times. For GDDR5, used in recent GPGPU systems, m is 3.

$p_i(t+1) = p_i(t) - 1$, if a request is scheduled from the ith queue of p and the request hits in the DRAM row buffer

$p_i(t+1) = p_i(t) - 1/m$, if a request is scheduled from ith queue of p and the request misses in the DRAM row buffer

$p_i(t+1) = p_i(t)$ otherwise

Queues for warps from core q are updated in the same manner.

Here, the best scheduling policy is to choose a $q_i(t)$ or $p_i(t)$ with a request with a row buffer hit to minimize $r(t+1)$; this policy becomes FR-FCFS.

B. Potential Function considering GPGPU Characteristics

Now, we extend the model to include two GPGPU characteristics: SIMD-execution, and tolerance. To reflect SIMD-execution, we introduce a new parameter, α, which is motivated by the α-MW(maximum weight) algorithm [7] in queuing systems.

$$r(t) = \sum_{i=1}^{N} (q_i(t)^{\alpha}) + \sum_{i=1}^{N} (p_i(t)^{\alpha}), \quad 0 < \alpha \leq 1 \tag{2}$$

α **Parameter:** The introduction of α serves two purposes, for $0 < \alpha < 1$, a policy based on the model will : (1) select a request from the warp with the shortest queue (assuming all queues have row hits) (2) try to schedule requests from the same warp. If the queue lengths do not change significantly between the scheduling of requests, (2) naturally follows from (1). α models the benefit of choosing a request from a warp with fewer requests. This benefit exists because a warp can resume execution only when all its requests are serviced. When α is 1, the model becomes the same as the one in section II-A. By having α less than 1, we can choose a memory request from a warp with fewer requests to minimize $r(t+1)$. When α is very close to 0, it is highly likely that requests from the warp with the fewest requests will be scheduled to minimize $r(t+1)$. Consequently, an α value close to 0 favors SJF and a value close to 1 favors FR-FCFS. The optimal value of α may be different for each benchmark - for benchmarks in which each warp has requests to only one row, any value of α between 0 and 1 would work, but for benchmarks whose warps have several requests to few rows, a small value of α may be more suitable. However, small values of α could result in frequent row-close and open commands being issued and reduce performance. Depending on the phases of long running warps in a benchmark, it may be prudent to vary the value of α dynamically for some benchmarks.

To reflect tolerance, we divide each term in the RHS of Equation (2) by the tolerance of the core to which the queue belongs. This way, queues from cores with lower tolerance contribute more to the potential than queues of same size from cores with higher tolerance. The final potential function which naturally models both tolerance and SIMD-execution is shown below.

$$r(t) = \sum_{i=1}^{N} (q_i(t)^{\alpha}) \frac{1}{ftol(c_q,t)} + \sum_{i=1}^{N} (p_i(t)^{\alpha}) \frac{1}{ftol(c_p,t)} \tag{3}$$

$$ftol(c_q,t) = r_q(t) \times wc \tag{4}$$

- $r_q(t)$: number of ready warps, i.e., warps that are not blocked on memory requests, in core q.
- wc: average compute work at time t that each (ready) warp in a core can perform without blocking. We assume that wc is constant.
- tolerance for core p is defined similarly.

III. Scheduling Policy

We call the scheduling policy we develop to minimize the potential function as α-SJF.

Tolerance Aware Core Selection: To keep the hardware simple, the scheduler can first choose the core with least tolerance, core C, and then select a request from core C to be serviced next. An alternate scheduling approach is for the scheduler to look at requests from all cores and find the request that minimizes $r(t+1)$ while considering tolerance as well. However, because of the complexity of the latter approach, we discard it.

Queue Selection: After a core is selected, we use Equation (2), albeit with queues from the selected core only, instead of Equation (3). At time t, we have to service the request that results in the minimum value of $r(t+1)$. This is the same as servicing the request that results in the maximum value of $(r(t) - r(t+1))$. Hence, using the first-order Taylor series approximation, α-SJF eventually chooses a request from queue i^* that satisfies Equation (5) where $1_A \in \{0,1\}$ is the indicator function of event row buffer hit (H) or miss (M).

$$i^* = \arg\min_i q_i(t)^{1-\alpha} \cdot 1_{i \in H} + m \cdot q_i(t)^{1-\alpha} \cdot 1_{i \in M}, \quad (5)$$

According to Equation (5), when queue i^* with a row buffer hit request is selected, the same queue i^* will be the optimal choice until the queue has no row buffer hits. When the queue has no more row buffer hits, we have to compute Equation (5) again to choose a queue (request). This computation is the same as choosing the minimum value between $q_j(t)^{1-\alpha}$ and $m \cdot q_k(t)^{1-\alpha}$. Here, $q_j(t)$ is the smallest queue with a row buffer hit request and $q_k(t)$ is the smallest queue without any row buffer hit requests. When all the queues are large, a queue with a row buffer hit request is selected, and when some (or all) of the queues are small, usually a request from the shortest queue is selected. The rationale behind this is that when all queues are large, it takes longer to empty the queues, so it might be better to choose a request that can be serviced in a short amount of time. However, when there are small queues, we can empty those queues fast and increase the number of ready warps in a short amount of time.

IV. Building α-SJF

Two factors are important for building α-SJF - (1) Storage to store the number of requests for each queue (warp) and (2) Computation to select the next request to be serviced. The amount of storage needed for $\alpha - SJF$ ultimately depends on the size of the per-bank request buffers, which is 128 in our experiments. In the extreme case, there could be 1 request each from 128 warps, this can be handled by having 128 5-bit counters for a total of 80B/bank.

The computation for Equation (5) can be simplified using the fact that x^n (for x > 0, n ≥ 0) will be minimum for the smallest value of x. We first identify the smallest queue with a rowhit, H_s, by comparing only the lengths of queues with rowhits. Similarly, we identify the smallest queue without a rowhit, M_s. In Equation (5), it is sufficient to compare only H_s and M_s. The comparison can be further simplified if we take the (1 - α)th root of the equation and store the precomputed value of $k = m^{(1/(1-\alpha))}$ in the system. If the length of H_s is more than k times the length of M_s, then M_s is selected, otherwise, H_s is selected.

V. Evaluation

A. Methodology

Simulations are done using MacSim [2], a cycle level GPGPU simulator. The simulated architecture, shown in Table I is based on NVIDIA's Fermi [3]. The baseline FR-FCFS policy gives equal priority to both reads and writes when new DRAM rows are opened, and once a row is open, it gives priority to reads with row hits and then writes with row hits. On average, our baseline performs better than a FR-FCFS policy that always prioritizes reads, since several GPGPU benchmarks generate significant write traffic.

To calculate the tolerance of a core, we approximate the number of ready warps with the number of warps assigned to each core (even completed warps are counted, but warps from retired thread blocks are not counted). This approximation reduces communication between the cores and the DRAM. The number of warps assigned to each core is sent to the DRAM at the start of the application and thereafter, only when it changes.

B. Results

Figure 1 shows the performance (IPC) of 6 benchmarks with two different flavors of α-SJF for different values of α relative to the baseline. While α-SJF

TABLE I
SIMULATED GPGPU ARCHITECTURE

Num. of cores	11
Front End	Fetch width: 1 warp-instruction/cycle, 4KB I-cache, stall on branch, 5 cycle decode
Execution core	2 warp-instructions/cycle, Frequency: 1.2 GHz; 8-wide SIMD, in-order scheduling; latencies are modeled according to the CUDA manual [4]
On-chip caches	16 KB software managed cache, 16 loads/2-cycle 1-cycle latency constant cache, 16 loads/2-cycle 1-cycle latency texture cache, 16 loads/2-cycle
DRAM	2 KB page, 16 banks with page interleaving, 128 Reqs/Bank, 8 channel, 102.6 GB/s, 1.6 GHz tCL=20, tRCD=28, tRP=12 (GDDR-5 [1])
Interconnection	20-cycle fixed latency

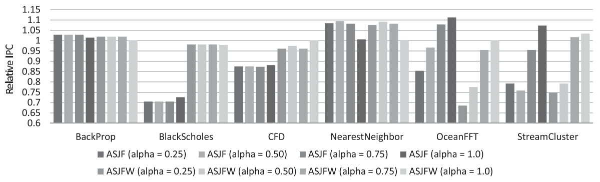

Fig. 1. IPC of α-SJF relative to FR-FCFS

services writes only after all reads are serviced, α-SJFW treats reads and writes equally. Also, α-SJFW assigns writes to the warp causing the write. [2]

Benchmarks (BackProp, NearestNeighbor, StreamCluster) whose warps send requests to one row and bank only, show benefit with at least one configuration of α-SJF or α-SJFW since such access patterns are exploited by SJF. OceanFFT primarily shows benefit because the write policy of α-SJF is to service writes only after servicing all reads. When α-SJF or α-SJFW provides benefit, larger values of α often show more benefit than smaller values (0.75 Vs. 0.25 or 0.5). Since smaller α values favor SJF, they issue more DRAM precharge and activate commands and reduce performance slightly. In the best case, BackProp, NearestNeighbor, OceanFFT and StreamCluster show benefit of 3%, 9%, 11% and 7%.

Warps in BlackScholes and CFD send requests to multiple rows in multiple banks, and for such benchmarks, α-SJF issues excessive precharge and activate commands, causing performance degradation. Because of short queues, α-SJF favors SJF, but since each warp sends requests to multiple rows, SJF causes rows to be closed even when there are potential row hit requests from other warps.

For complex access patterns, there should be co-ordination between banks. However fine grained co-ordination that may be necessary to schedule requests from one warp to different banks in parallel may be impractical. Also, the potential function has to be updated to account the cost of serving row miss requests more accurately. In the current model it is assumed that serving a row miss over a row hit affects the value of the potential at time t+1 only, but serving a row miss request at time t means no requests can be scheduled at t+1 and t+2.

Note that performance with α = 1.0 is not the same as the baseline since α-SJF first chooses a core and then chooses a request from the selected core. On the other hand, the baseline FR-FCFS usually chooses the oldest request with a row hit. If α-SJF is implemented without the core-selection step, then it produces the same results as FR-FCFS for α= 1.0.

VI. CONCLUSION

We present a GPGPU characteristic-aware DRAM scheduling policy, α-SJF, that is motivated by a Lyapunov-Foster potential function. Evaluations show that α-SJF improves several benchmarks, but not benchmarks such as BlackScholes and CFD, whose warps send requests to multiple rows and/or banks. In future work, we will overcome the limitations of our model and develop more enhanced scheduling policies. We will also improve the model to identify the best α value at run-time or based on application characteristics. We believe that α-SJF policy and our cost model open a new research direction in DRAM scheduling.

[2]For GPGPU applications with excessive writes, the way writes are handled can alter performance. In our future work we will explore more on write scheduling policies.

REFERENCES

[1] "HYNIX GDDR5 SGRAM," http://www.hynix.co.kr/inc/pdfDownload.jsp? path=/datasheet/pdf/graphics/H5GQ1H24AFR(Rev1.0).pdf.
[2] "MacSim," http://code.google.com/p/macsim/.
[3] NVIDIA, "Fermi: Nvidia's next generation cuda compute architecture," http://www.nvidia.com/fermi.
[4] *CUDA Programming Guide, V4.0*, NVIDIA Corporation.
[5] S. Rixner, "Memory controller optimizations for web servers," in *MICRO-37*. Los Alamitos, CA, USA: IEEE Computer Society, 2004, pp. 355–366.
[6] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *ISCA-27*. Washington, DC, USA: IEEE Computer Society, 2000.
[7] D. Shah and D. Wischik, "Switched networks with maximum weight policies: Fluid approximation and multiplicative state space collapse," *CoRR*, vol. abs/1004.1995, 2010.
[8] W. K. Zuravleff and T. Robinson, "Controller for a synchronous dram that maximizes throughput by allowing memory requests and commands to be issued out of order," U.S. Patent Number 5,630,096, May 1997.