Springer

Dear Author,

Here are the proofs of your article.

- You can submit your corrections **online**, via **e-mail** or by **fax**.
- For **online** submission please insert your corrections in the online correction form. Always indicate the line number to which the correction refers.
- You can also insert your corrections in the proof PDF and **email** the annotated PDF.
- For fax submission, please ensure that your corrections are clearly legible. Use a fine black pen and write the correction in the margin, not too close to the edge of the page.
- Remember to note the **journal title**, **article number**, and **your name** when sending your response via e-mail or fax.
- **Check** the metadata sheet to make sure that the header information, especially author names and the corresponding affiliations are correctly shown.
- **Check** the questions that may have arisen during copy editing and insert your answers/ corrections.
- **Check** that the text is complete and that all figures, tables and their legends are included. Also check the accuracy of special characters, equations, and electronic supplementary material if applicable. If necessary refer to the *Edited manuscript*.
- The publication of inaccurate data such as dosages and units can have serious consequences. Please take particular care that all such details are correct.
- Please **do not** make changes that involve only matters of style. We have generally introduced forms that follow the journal's style.
  Substantial changes in content, e.g., new results, corrected values, title and authorship are not allowed without the approval of the responsible editor. In such a case, please contact the Editorial Office and return his/her consent together with the proof.
- If we do not receive your corrections **within 48 hours**, we will send you a reminder.
- Your article will be published **Online First** approximately one week after receipt of your corrected proofs. This is the **official first publication** citable with the DOI. **Further changes are, therefore, not possible.**
- The **printed version** will follow in a forthcoming issue.

Please note

After online publication, subscribers (personal/institutional) to this journal will have access to the complete article via the DOI using the URL: http://dx.doi.org/[DOI].
If you would like to know when your article has been published online, take advantage of our free alert service. For registration and further information go to: http://www.link.springer.com.

Due to the electronic nature of the procedure, the manuscript and the original figures will only be returned to you on special request. When you return your corrections, please inform us if you would like to have these documents returned.

Metadata of the article that will be visualized in OnlineFirst

Please note: Images will appear in color online but will be printed in black and white.

ArticleTitle	A Credit-Based Load-Balance-Aware CTA Scheduling Optimization Scheme in GPGPU	
Article Sub-Title		
Article CopyRight	Springer Science+Business Media New York (This will be the copyright line in the final PDF)	
Journal Name	International Journal of Parallel Programming	
Corresponding Author	Family Name	**Guo**
	Particle	
	Given Name	**He**
	Suffix	
	Division	School of Software Technology
	Organization	Dalian University of Technology
	Address	Dalian, China
	Email	guohe@dlut.edu.cn
Author	Family Name	**Yu**
	Particle	
	Given Name	**Yulong**
	Suffix	
	Division	School of Software Technology
	Organization	Dalian University of Technology
	Address	Dalian, China
	Division	Department of Electrical and Computer Engineering
	Organization	Virginia Commonwealth University
	Address	Richmond, VA, USA
	Email	yuylong@acm.org
Author	Family Name	**He**
	Particle	
	Given Name	**Xubin**
	Suffix	
	Division	Department of Electrical and Computer Engineering
	Organization	Virginia Commonwealth University
	Address	Richmond, VA, USA
	Email	xhe2@vcu.edu
Author	Family Name	**Wang**
	Particle	
	Given Name	**Yuxin**
	Suffix	
	Division	School of Computer Science and Technology
	Organization	Dalian University of Technology
	Address	Dalian, China
	Email	wyx@dlut.edu.cn
Author	Family Name	**Chen**

	Particle	
	Given Name	**Xin**
	Suffix	
	Division	School of Software Technology
	Organization	Dalian University of Technology
	Address	Dalian, China
	Email	chenx_dlut@163.com

Schedule	Received	8 July 2014
	Revised	
	Accepted	31 July 2014

Abstract	GPGPU improves the computing performance due to the massive parallelism. The cooperative-thread-array (CTA) schedulers employed by the current GPGPUs greedily issue CTAs to GPU cores as soon as the resources become available for higher thread level parallelism. Due to the locality consideration in the memory controller, the CTA execution time varies in different cores, and thus it leads to a load imbalance of the CTA issuance among the cores. The load imbalance causes the computing resources under-utilized, and leaves an opportunity for further performance improvement. However, existing warp and CTA scheduling policies did not take account of load balance. We propose a credit-based load-balance-aware CTA scheduling optimization scheme (CLASO) piggybacked to a standard GPGPU scheduling system. CLASO uses credits to limit the amount of CTAs issued on each core to avoid the greedy issuance to faster executing cores as well as the starvation to leftover cores. In addition, CLASO employs the global credits and two tuning parameters, active levels and loose levels, to enhance the load balance and the robustness. Instead of a standalone scheduling policy, CLASO is compatible with existing CTA and warp schedulers. The experiments conducted using several paradigmatic benchmarks illustrate that CLASO effectively improves the load balance by reducing 52.4 % idle cycles on average, and achieves up to 26.6 % speedup compared to the GPGPU baseline scheduling policy.

Keywords (separated by '-')	GPGPU - CTA Scheduler - Credit-based load-balance-aware scheduling scheme - Load Balance
Footnote Information	

Int J Parallel Prog
DOI 10.1007/s10766-014-0318-5

A Credit-Based Load-Balance-Aware CTA Scheduling Optimization Scheme in GPGPU

Yulong Yu · Xubin He · He Guo · Yuxin Wang · Xin Chen

Received: 8 July 2014 / Accepted: 31 July 2014

Abstract GPGPU improves the computing performance due to the massive parallelism. The cooperative-thread-array (CTA) schedulers employed by the current GPGPUs greedily issue CTAs to GPU cores as soon as the resources become available for higher thread level parallelism. Due to the locality consideration in the memory controller, the CTA execution time varies in different cores, and thus it leads to a load imbalance of the CTA issuance among the cores. The load imbalance causes the computing resources under-utilized, and leaves an opportunity for further performance improvement. However, existing warp and CTA scheduling policies did not take account of load balance. We propose a credit-based load-balance-aware CTA scheduling optimization scheme (CLASO) piggybacked to a standard GPGPU scheduling system. CLASO uses credits to limit the amount of CTAs issued on each core to avoid the

Y. Yu · H. Guo (✉) · X. Chen
School of Software Technology, Dalian University of Technology, Dalian, China
e-mail: guohe@dlut.edu.cn

Y. Yu
e-mail: yuylong@acm.org

X. Chen
e-mail: chenx_dlut@163.com

Y. Yu · X. He
Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA, USA
e-mail: yyu2@vcu.edu

X. He
e-mail: xhe2@vcu.edu

Y. Wang
School of Computer Science and Technology, Dalian University of Technology, Dalian, China
e-mail: wyx@dlut.edu.cn

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

greedy issuance to faster executing cores as well as the starvation to leftover cores. In addition, CLASO employs the global credits and two tuning parameters, active levels and loose levels, to enhance the load balance and the robustness. Instead of a standalone scheduling policy, CLASO is compatible with existing CTA and warp schedulers. The experiments conducted using several paradigmatic benchmarks illustrate that CLASO effectively improves the load balance by reducing 52.4 % idle cycles on average, and achieves up to 26.6 % speedup compared to the GPGPU baseline scheduling policy.

Keywords GPGPU · CTA scheduler · Credit-based load-balance-aware scheduling scheme · Load balance

1 Introduction

GPGPU improves the computing performance due to its massive parallelism. The schedulers in GPGPU are essential to achieve this high performance. An efficient scheduler should avoid the idle cycles appearing on all the functional units, and keeps the hardware resources as busy as possible.

There are two layers of schedulers in current GPGPUs, which are cooperative-thread-array (CTA) scheduler and warp scheduler. The baseline CTA schedulers only focus on the thread level parallelism (TLP) and greedily issue a CTA to a core as soon as whose resources become available. The memory controller usually prioritizes the neighboring-addressed data accesses for locality, and varies the CTA execution time between different cores. It leads the CTA schedulers to issue too many CTAs to the faster executing cores that finish their previous running CTAs earlier than other cores, but starve the leftover cores. It results in a load imbalance, especially when the number of CTAs in the kernel is indivisible by the concurrent CTA capacity. Existing warp and CTA scheduling policies such as Two-Level warp scheduler (TL) [1], CTA-aware warp scheduler (OWL) [2], and Dynamic CTA scheduling mechanism (DYNCTA) [3] were proposed to improve TLP, cache and memory access efficiency, and memory bank level parallelism, but did not take account of the CTA load imbalance.

In this paper, we propose a credit-based load-balance-aware CTA scheduling optimization scheme (CLASO) that targets to improve the CTA load balance. Rather than greedily issuing CTAs as soon as the resources become available, CLASO limits the amount of CTAs issued on each core. CLASO allocates credits to each core when a GPU kernel is launched, and every CTA issuance consumes a credit. CLASO only allows the CTA issuance request when there are remaining credits in the corresponding core. In addition to the above local credits, CLASO employs the global credits and two tuning parameters, active levels and loose levels, to enhance the load balance and the robustness. CLASO is not a standalone scheduling policy. It works as a module piggybacked to a standard GPGPU scheduling system, so it is compatible with existing CTA and warp schedulers. CLASO keeps the underlying scheduling policy untouched as much as possible, and further improves the load balance and the performance. Our experimental results show that CLASO decreases 52.4 % load-imbalanced idle cycles on average, and achieves up to 26.6 % speedup upon the baseline GPGPU sched-

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

uler. The experiments also illustrate the effectiveness of CLASO with other existing scheduling policies, such as TL, OWL, and DYNCTA.

The contributions in our work are two-fold.

1. We investigate the load imbalance problem of the current GPGPU scheduling system and give a detailed analysis of this problem.
2. We propose a scheduling optimization scheme named CLASO for the CTA issuance load balance. Our approach is compatible with existing warp and CTA schedulers.

The rest of this paper is organized as follows. The GPGPU baseline architecture and the load imbalance problem are discussed in Sect. 2. Section 3 describes the design of CLASO. The experimental methodology and results are illustrated in Sect. 4. Section 5 summarizes related work. Finally we conclude this paper in Sect. 6.

2 Background and Motivation

2.1 GPGPU Baseline Architecture

The current GPGPUs as illustrated in Fig. 1a is composed of a series of cores with parallel execution resources such as functional units, registers. A GPGPU core executes instructions in a single-instruction-multi-thread (SIMT) way, where a group of threads called warps (usually 32 threads per warp) executes simultaneously on respective processing lanes. Cores have their own L1 caches, but the accesses to L2 caches and DRAMs have to go through the interconnect on-chip network. DRAM is organized into banks for parallel memory accesses. The memory accesses to the same bank have to be operated serially, but the accesses to the same row will be prioritized.

Applications employing the power of GPGPU are usually written by the parallel programming interfaces such as CUDA [4] and OpenCL [5]. They organize the GPGPU computing tasks as kernels. A GPU kernel as illustrated in Fig. 1b is defined by a massive amount of threads. These threads are tiled into coarse-grain groups referred as blocks or CTAs. The GPU core accepts compute tasks at the granularity of CTA, and multiple CTAs can be concurrently placed in the same core for a high TLP.

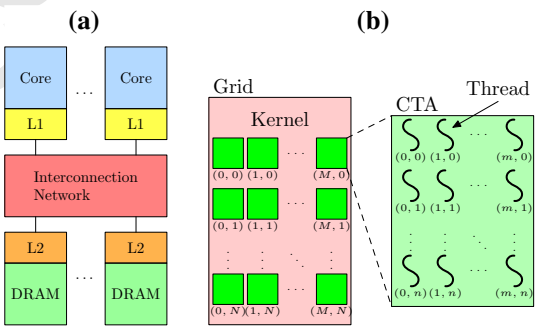

Fig. 1 GPGPU baseline architecture and programming model. **a** Architecture, **b** programming model

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

The GPGPU schedulers need give reasonable placement and execution order of CTA and warps of the kernels launched on GPGPU. There are two layers of schedulers inside current GPGPU implementation.

1. A *CTA Scheduler* is the upper-layer scheduler that decides when and where to issue a CTA. In the baseline architecture, the CTA scheduler uses a greedy strategy. Whenever resources become available, it selects an unissued CTA and issues it to the corresponding core.
2. A *Warp Scheduler* is the lower-layer scheduler equipped in every GPU core. The issued CTAs are reorganized as warps. The warp scheduler decides when and which warp to execute an instruction. Loose round robin (LRR) scheduling policy is commonly used.

2.2 CTA Load Imbalance

The baseline CTA scheduling policy does not guarantee the load balance. We execute 4 benchmarks on a GeForce GTX 480 with 15 GPU cores to test the CTA load balance. The description of the benchmarks used in this subsection can be found in Sect. 4.1. We execute each benchmark using different grid sizes, i.e. the number of CTAs, and record the average ratios of idle cycles on all the GPU cores using NVIDIA Visual Profiler `nvvp` [6] as illustrated in Fig. 2. An idle cycle is a clock cycle when a core has no CTA to execute. In a load-balance-aware CTA scheduler, the result curve shall present a sawtooth wave whose period equals to the number of GPU cores, and the amplitude shall be small. However, the experimental results show a different pattern. The slumps appear when the grid sizes are the multiples of the concurrent CTA capacity (The concurrent CTA capacities change according to the kernel's resource occupancy. [4]). The amplitudes are large, and the highest peaks achieve nearly 20 % that imply significant resource under-utilization. Therefore, we speculate that the load imbalance is a key factor for the results.

We use GPGPU-Sim [8], a cycle-accurate GPGPU simulator, to understand the kernel execution in detail. The experimental setup is described in Sect. 4. We select a benchmark SP [7]. Figure 3a shows the executed instructions per 500 cycles. About

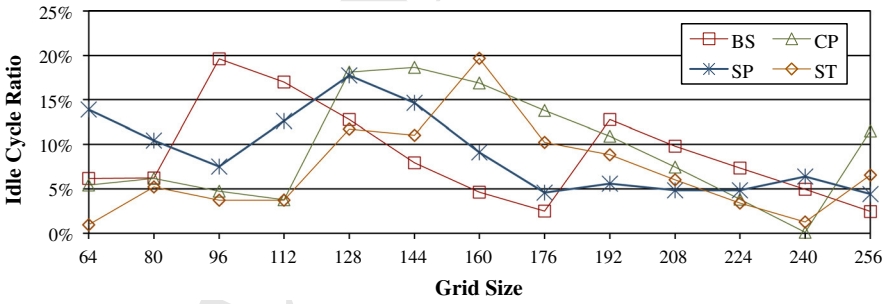

Fig. 2 The percentages of the idle cycles in different grid sizes. An idle cycle is a cycle that a core has no CTA to execute. The long periods of idle cycle variance along the grid sizes and the large amplitudes between the peaks and the slumps present the load imbalance in CTA issuance

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

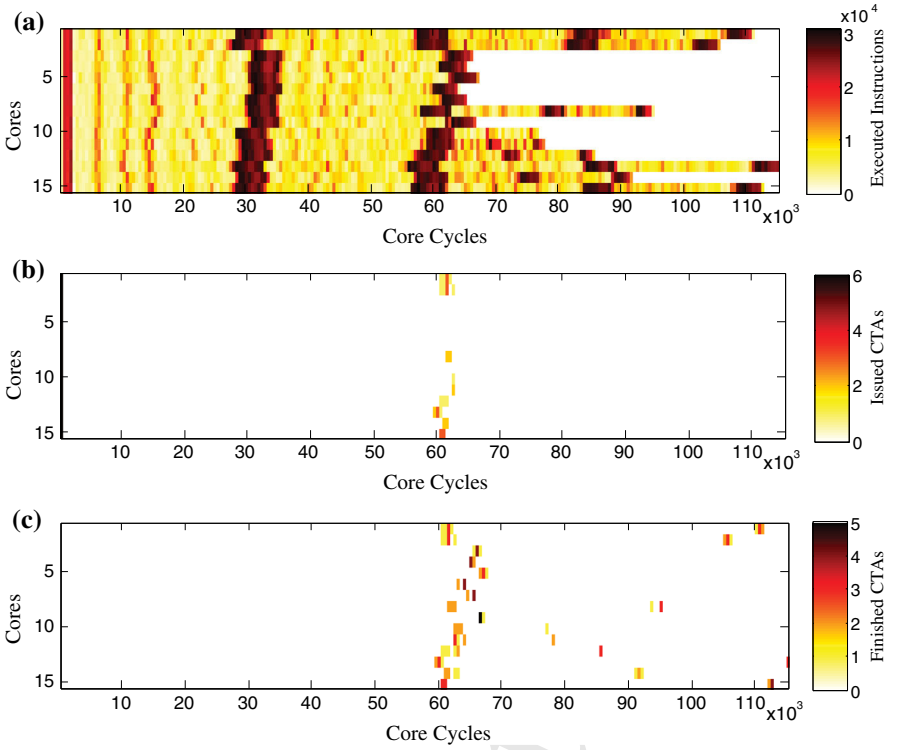

Fig. 3 The detailed per-core profiling data of SP [7] from GPGPU-Sim [8]. The executed instructions per 500 cycles (Fig. 3a) demonstrate that the baseline CTA scheduler suffers from the load imbalance. Figure 3c presents that the CTAs inside the same core have similar execution time, but the CTA execution time is different between different cores. Therefore, the baseline scheduler issues too many CTAs to the faster executing cores (e.g. core 15), but starves the leftover cores (e.g. core 5) as shown in Fig. 3b. **a** The executed instructions, **b** The issued CTAs, **c** The finished CTAs

half of the cores remain idle in the second half of execution period due to the load imbalance. Figure 3b, c present the numbers of CTAs issued and finished on each core. The CTAs issued on the same core usually have more similar execution time than those issued on different cores, so a core finishes all its current set of executing CTAs during a very short period. Therefore, CTA scheduler issues too many CTAs to the faster executing cores (e.g. core 15), but the leftover cores are starved because all CTAs have been issued resulting in no CTAs for them (e.g. core 5).

All the GPGPU cores execute identical codes following the same clock. The only thing that varies the execution steps among cores is the shared control logics. The memory controller is a shared control logic that significantly affects the performance. We test the memory access balance among cores using several benchmarks. Figure 4 shows the average difference ratios of the processed memory accesses of 10 random selected time intervals. A difference ratio is calculated by the fraction between the maximum count and the minimum counts of the processed memory accesses among all the cores. A core-balanced memory access scheduling will present a difference

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

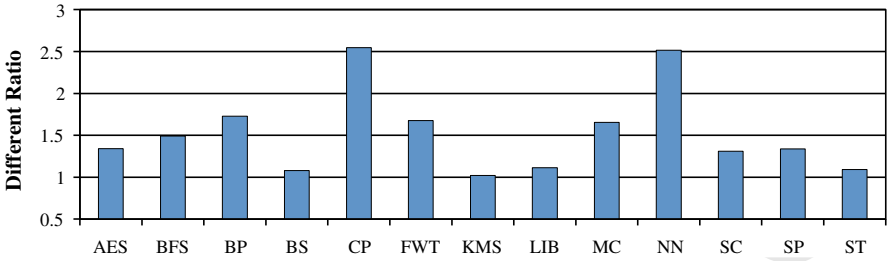

Fig. 4 The average difference ratios of the processed memory accesses of 10 random selected sampling intervals. The difference ratio is calculated from the fraction between the maximum count and the minimum count of the processed memory accesses among all the cores. Most of these benchmarks have poor GPU-core-balanced memory accesses

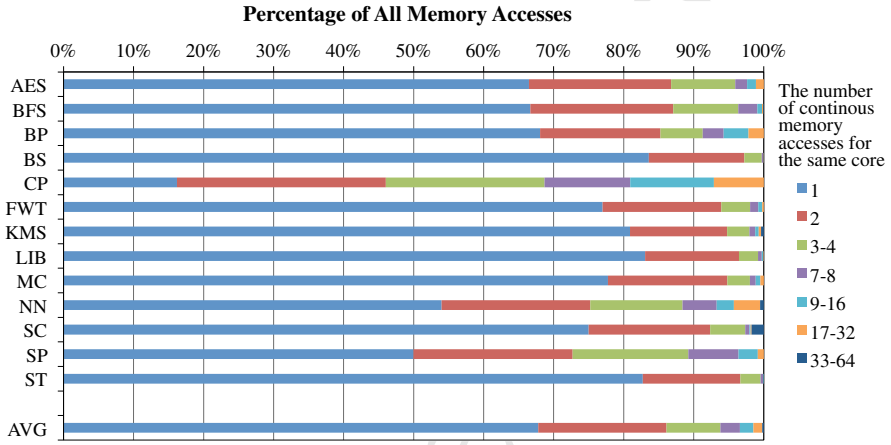

Fig. 5 The percentages of the continuous processed memory accesses for the same core. More than 30 % on average of the continuous processed memory accesses are served for the same core, which makes the the memory access load imbalance among cores, and finally leads to the CTA issuance load imbalance

ratio near to 1, but the experimental results of most of these benchmarks are larger than 1.2, where CP and NN present the highest difference ratios, over 2.5. They exhibit the memory accesses are core-imbalanced.

We test the memory scheduling decision for GPU cores to find out the reason of the imbalance. As illustrated in Fig. 5, more than 30 % of memory accesses on average are processed followed by another for the same core. Current memory controller prioritizes the accesses that belong to the same address row, i.e. the memory accesses to neighboring addresses are usually prioritized. The memory requests from the same core usually present stronger locality. Therefore, the memory controller usually prioritizes the memory accesses from part of cores in a certain period. It causes the core-imbalanced memory accesses, and further makes the CTA execution time differences between cores. Finally, the greedy CTA scheduling policy issues too many CTAs to the faster executing core, and leads to the CTA load imbalance.

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

The goal to avoid the CTA load imbalance is to issue similar amount of CTAs on each GPU core. Two alternative methods can be adopted. The first is to propose a core-aware memory scheduling policy to avoid the core-imbalanced memory accesses. However, it has to degrade the data locality and increase the row reopen operations in memory controllers, which affects the memory efficiency and the performance. The Second is to propose a CTA scheduling optimization scheme to avoid the excessive CTA issuance and the CTA starvation. This kind of method does not affect the memory operations, which bypasses the risk of performance degradation. Additionally, the CTA scheduling scheme is built directly to our goal. Following this motivation, we propose a load-balance-aware CTA scheduling optimization scheme that is described in the next section.

3 A Credit-Based Load-Balance-Aware Scheduling Optimization Scheme (CLASO)

We propose a credit-based load-balance-aware CTA scheduling optimization scheme (CLASO). We highlight the compatibility of CLASO with underlying schedulers instead of a standalone scheduling policy. This section gives the detailed design of CLASO (Subsect. 3.1) along with a case study (Subsect. 3.2), as well as the hardware overhead analysis (Subsect. 3.3).

3.1 CLASO Design

CLASO works as a module piggybacked to a standard GPGPU scheduling system as illustrated in Fig. 6. The standalone scheduling policies, such as LRR and TL, are hard to combine together into a super solution, so it has to statically select one among them. CLASO gives a solution that cooperates with existing or future proposed CTA and warp scheduling policies. When the underlying CTA scheduler generates a CTA issuance request, the CLASO module sends a decision to the scheduler to allow or refuse this CTA issuance. The CTA load balance is guaranteed by CLASO via this decision. In usual cases, CLASO does not interfere the decision by the underlying scheduling policies in the CTA and warp schedulers. Only when the excessive CTA

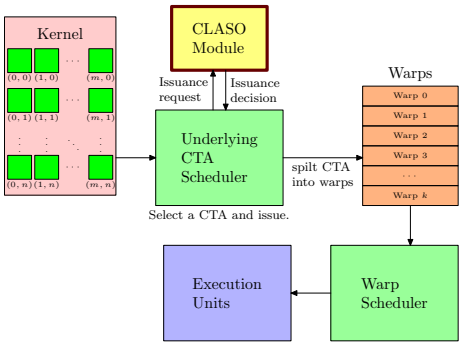

Fig. 6 The GPGPU scheduling system architecture piggybacking with CLASO. CLASO works as a module in the GPGPU scheduling system. The CTA scheduler needs to send a request to CLASO module before a CTA issuance. CLASO makes the decision whether the CTA issuance is allowed according to the execution states

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

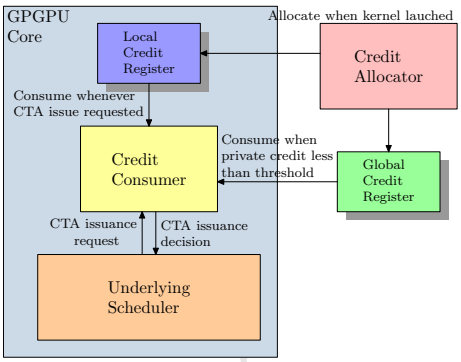

Fig. 7 The structure of CLASO. The Credit Allocator allocates credits to the local credit registers and the global credit register when a kernel launched. The Credit Consumer consumes a credit when a new CTA issuance request is received. The exhaustion of credits indicates that too many CTAs have been issued on the corresponding core, then the Credit Consumer refuses the further CTA issuance requests on this core to guarantee the load balance

issuance appears, CLASO holds the CTA issuance requests, and helps the underlying scheduler making a better decision by achieving load balance.

CLASO module uses the credits allocated to GPU cores to control the total number of CTAs that can be issued on each core. The structure of CLASO is designed as Fig. 7, which contains two main functional units named *Credit Allocator* and *Credit Consumer*. The credit allocator allocates credits when a kernel is launched on the GPGPU. The credit consumer in each core consumes a credit when the underlying CTA scheduler sends a CTA issuance request, and makes a decision whether the request is allowed or refused. Note that the number of total allocated credits must be a multiple of the number of cores in the GPGPU. This is too coarse grained to achieve CTA load balance, and will still cause a starvation to the slower executing cores. Therefore, we further introduce *Global Credits* shared among cores to cooperate with the credits allocated for each core. We refer the credits inside each core as *Local Credits*. When the local credits are fewer than a threshold, the corresponding CTA issuance request is allowed only if there is a global credit can be also consumed with the local credit.

The numbers of local credits and global credits allocated are calculated from the CTA numbers in the kernel. In the tightest limit on CTA issuance, all the cores issue almost the same amount of CTAs of a kernel, and the maximum difference among cores is at most one CTA. Therefore, in this case for a kernel with n_{cta} CTAs, the credit allocator allocates $\left\lceil \frac{n_{\text{cta}}}{n_{\text{core}}} \right\rceil$ local credits per core and $((n_{\text{cta}} - 1) \mod n_{\text{core}} + 1)$ shared global credits, where n_{core} is the number of GPU cores. The threshold where a local credit must be consumed with a global credit is tuned by two parameters named as Active Levels and Loose Levels as illustrated in Fig. 8. They are used to loose the tightest limit, and give a robustness to tolerate the diverse execution time between CTAs due to the program logic. Active Levels p_A indicate the layers of global credits. Each layer contains n_{core} credits except the last layer that contains $((n_{\text{cta}} - 1) \mod n_{\text{core}} + 1)$ global credits. Loose Levels p_L are the extra local credits allocated to each core, i.e. the orange-colored credits in Fig. 8. If the remaining local credits in a core are less than $(p_A + p_L)$, a global credit must be consumed along with a local credit. The tightest limit are the lowest-bound case that has only one layer of global credits and no extra local credits, so we denote it as $(p_A = 1, p_L = 0)$. Small p_A and p_L bring tighter limit and better static CTA load balance among GPU cores, but

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

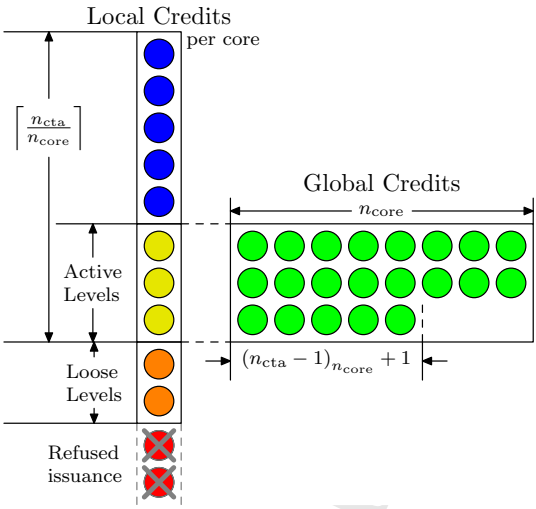

Fig. 8 The credit allocation policy in CLASO. The local credits are allocated to every core, and the shared global credits are used to complement the local credits. The CTA issuance request can be directly allowed when consuming a *blue-colored* local credit. A global credit must be consumed when the credit consumer encounters a *yellow-* or *orange-colored* local credit. In this case, the CTA issuance request will be refused when the global credits are exhausted. Active and Loose Levels are used to tune the allocation policy for the robustness

Algorithm 1 Credit-based Load-balance-aware Scheduling Scheme

▷ $p_A \geq 1$ denotes the Active Levels;
▷ $p_L \geq 0$ denotes the Loose Levels;
▷ n_{cta} denotes the number of CTAs in the launched kernel;
▷ n_{core} denotes the number of cores in GPGPU;
▷ $C_p\,[c]$ denotes the local credits of core c;
▷ C_g denotes the global credits;

if a new kernel is launched **then**
  $C_p\,[c] \leftarrow \left\lfloor \frac{n_{\text{cta}}}{n_{\text{core}}} \right\rfloor + p_L$ for $\forall c$ in GPGPU;
  $C_g \leftarrow ((n_{\text{cta}} - 1) \mod n_{\text{core}}) + 1 + (p_A - 1) \cdot n_{\text{core}}$;
end if
for all core c in GPGPU **do**
  if There exists a CTA issuance request in c **then**
    $C_p\,[c] \leftarrow C_p\,[c] - 1$;
    if $C_p\,[c] \geq p_A + p_L$ **then**
      Allow the CTA issuance request;
    else if $C_p\,[c] \geq 0$ **then**
      $C_g \leftarrow C_g - 1$;
      if $C_g \geq 0$ **then**
        Allow the CTA issuance request;
      else
        Refuse the CTA issuance request;
      end if
    else
      Refuse the CTA issuance request;
    end if
  end if
end for

less robustness with the dynamic execution time difference between CTAs. However, larger p_A and p_L issue more CTAs to the faster executing cores, and degrade the static load balance. CLASO will degenerate to the baseline scheduler with too large p_A and p_L. According to our experimental results described in Sect. 4.4, the best choices of the tuning parameters are ($p_A = 1$, $p_L = 0$) or ($p_A = 2$, $p_L = 0$).

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

The pseudo-code of the credit allocation and consuming policy is given in Algorithm 1, where Lines 1–4 are the logic of the credit allocator, and Lines 5–21 are the logic of the credit consumer. Note that CLASO refuses the CTA issuance request only when the credits are exhausted. The decision on the TLP by underlying scheduler is not affected until a load imbalance occurs. Therefore, CLASO maintains the TLP as much as possible and effectively cooperates with the underlying schedulers.

3.2 A Case Study

We take an simple example as a case study. Assuming a kernel with 17 CTAs runs on a 4-core GPGPU, and each core can issue at most 3 concurrent CTAs. The reasons why we choose this configuration are two-fold. First, when we zoom out to a real GPGPU with 15–30 cores and 8 concurrent CTAs per core, the corresponding grid size in the same scale is about 256 a common-used case in a GPGPU kernel. Second, this scenario is simple, thus it is easy to demonstrate CLASO clearly.

Figure 9 demonstrates the scheduling decisions by the baseline scheduler. At the beginning of the kernel execution, the CTA scheduler issues three CTAs on each core in round robin to maximize TLP. When the fastest executing core (core 3) finishes its first set of three CTAs (CTA3, CTA7, and CTA11), the CTA scheduler issues another three CTAs (CTA12, CTA13, and CTA14) to it, while other cores do not finish their first set of three CTAs at this time. The same thing happens to core 0, and the last two CTAs (CTA15 and CTA16) are issued on it. Finally, when cores 1 and 2 finish their first set of three CTAs, there are no more CTAs to issue. These two cores have to be idle in the second half of the execution period.

Figure 10 illustrates the scheduling decisions by CLASO. We choose the tuning parameters as $(p_A = 1, p_L = 0)$ in this example. First, the credit allocator allocates $\left\lceil \frac{n_{\text{cta}}}{n_{\text{core}}} \right\rceil + p_L = 5$ local credits to each core and $((n_{\text{cta}} - 1) \mod n_{\text{core}}) + 1 + (p_A - 1) \cdot n_{\text{core}} = 1$ global credit. At the beginning of the execution, the CTA scheduler issues

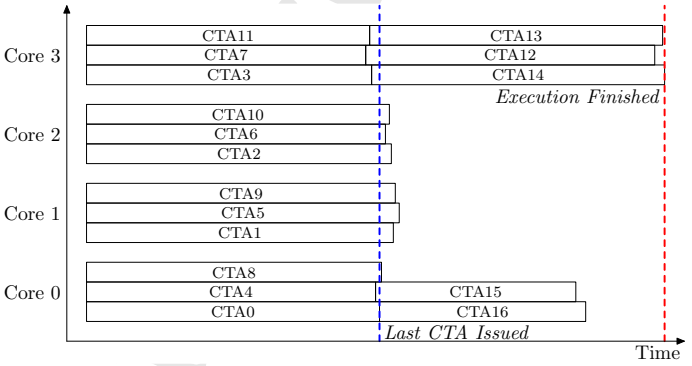

Fig. 9 The example of CTA load imbalance in the baseline scheduler. The varieties of the CTA execution time between different cores make the scheduler issuing too many CTAs to the faster executing cores such as core 0 and 3, while core 1 and 2 have no opportunity to run a further CTA in the second half of execution period

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

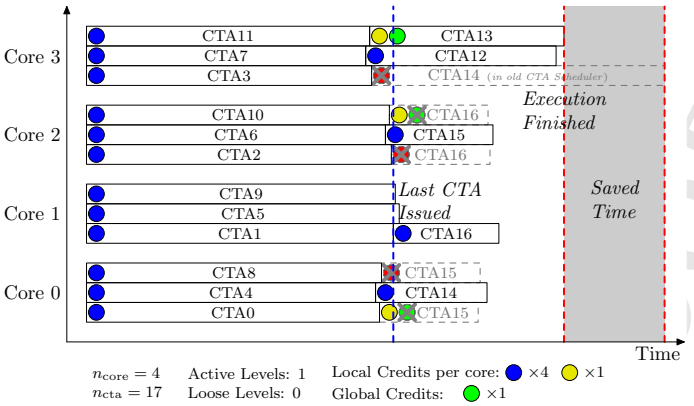

Fig. 10 The CLASO scheduling decision for the same example in Fig. 9. The local and global credits limit the amount of issuable CTAs on each core. A CTA load balance is achieved by CLASO

three CTAs on each core, and the credit consumer consumes 3 local credits of each core respectively. CTA7 on core 3 first finishes its execution. CTA scheduler sends a CTA issuance request to the credit consumer. After consuming a local credit of core 3, the credit consumer allows the request. The CTA scheduler selects and issues an unissued CTA (CTA12) on core 3. Shortly after that, CTA11 also finishes execution on core 3, and the CTA scheduler sends another request. The credit consumer consumes the last local credit (yellow-colored credit in Fig. 10) on core 3 with a global credit (green-colored credit in Fig. 10), and allows this request to issue CTA13 on core 3. When CTA3 finishes execution on core 3, the CTA issuance request is refused because there is no local credit for core 3. Next, CTA4 finishes execution on core 0, the credit consumer allows a new CTA (CTA14) issuance request by consuming a local credit. However, when CTA0 finishes on core 0, the credit consumer consumes the last local credit in core 0. However, the exhausted global credit makes the CTA issuance request refused. After that, another CTA issuance request from core 0 is also refused because of the lack of local credit. The same thing happens to cores 1 and 2, and there is one CTA executing on each of these three cores in the second half of execution period. CLASO achieves a CTA load balance.

Imagine that there is no global credit in CLASO. When core 0 tries to issue CTA15 when CTA0 is finished, the issuance request will be allowed because there is still one local credit of core 0. It will result in the starvation of core 1, because there is no unissued CTA when CTA1 is finished. The absence of global credit causes under-satisfied scheduling compared to our current design.

3.3 Hardware Overhead

CLASO has a low hardware overhead. The credit consumer only performs subtract and compare operations. The credit allocator performs division and mod computations, but the fixed divisor leads to a low overhead. Additionally, the global credit register needs

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

an arbitrator to guarantee the atomic operation. These functional units only consume a very small die area.

CLASO requires a small amount of registers. For a low-end graphic chip without the multi-concurrent-kernel support, CLASO requires $(n_{\text{core}} + 1)$ registers to hold the local and global credits. For a high-end device that supports the concurrent kernel execution, CLASO requires one set of credit registers for each concurrent kernel, i.e. totally $N_{\text{ker}} \cdot (n_{\text{core}} + 1)$ registers, where N_{ker} denotes the concurrent kernel capacity. These numbers are only negligible fractions compared to the total registers in the current generation of GPGPUs.

4 Experiments and Results

We simulate CLASO using GPGPU-Sim [8], and conduct experiments to evaluate its effectiveness on speedup and load balance, as well as the TLP effect when CLASO is used with LRR, TL, OWL and DYNCTA. The sensitivity of tuning parameters are also discussed in this section.

4.1 Methodology

The CLASO is simulated and the experiments are conducted in a widely used GPGPU performance simulator GPGPU-Sim 3.2.0 [8] with the default configuration of NVIDIA Tesla C2050 [9] listed in Table 1. GPGPU-Sim simulates every cycle during the kernel execution, and outputs the insight profiling data that helps understanding the details of the CLASO design.

We use 13 paradigmatic benchmarks listed in Table 2 that suffer from the performance penalty caused by the CTA load imbalance. These benchmarks are tested upon four existing warp and CTA scheduling policies, LRR, TL, OWL, and DYNCTA. LRR [8] is a commonly used but simple warp scheduling policies, which issues instructions from consecutive warps one by one. TL [1] targets to maximize the overlap of the memory access and the computation. OWL [2] is an upgrade of TL with the warp preemption mechanism to improve the cache efficiency. DYNCTA [3] is a CTA scheduler that dynamic changing the number of executing CTAs on each core. We choose OWL and DYNCTA because they diversify the execution time between CTAs inside a same core that may weaken CLASO's efficiency.

We use the issued instructions per cycle (IPC) as the metric of performance. The speedup is the normalized IPC of CLASO relative to the corresponding baseline scheduler. The load imbalance is quantized by the normalized idle core cycles to the baseline scheduler. The TLP efficiency is measured by the scoreboard conflicts, the number of cycles when the warp scheduler cannot issue a new instruction because it waits the compute results from its previous instructions. High TLP efficiency leads to less scoreboard conflicts because the conflicts are hidden if more warps execute concurrently.

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

Table 1 GPGPU-Sim configuration

# Cores	14
Warp size	32
SIMD pipeline width	16
# Threads/Core	1,024
# Registers/Core	32,768
Shared memory/Core	48 KB
Constant cache/Core	8 KB, 64B line, 2-way assoc. LRU
Texture cache/Core	12 KB, 128B line, 24-way assoc. LRU
# Memory partitions	6
L1 Data cache	16 KB, 128B line, 4-way assoc. LRU
L2 Unified cache	128 KB / Memory partition, 128B line, 16-way assoc. LRU
Core clock	1,150 MHz
Interconnect clock	1,150 MHz
Memory clock	750 MHz
DRAM request queue size	16
Memory scheduler	Out of order (FR-FCFS)
DRAM memory timing	$t_{CL} = 12, t_{RP} = 12, t_{RC} = 40, t_{RAS} = 28, t_{RCD} = 12, t_{RRD} = 6$
DRAM bus width	384 bits

Table 2 Benchmarks used in experiments

Abbr.	Src.	Applications	#K	#C	CS	CpC
AES	Sim [8]	AES cryptography	1	257	256	6
BFS	Rodinia [10]	Breadth first search	20	2,560	512	3
BP	Rodinia [10]	Back propagation	2	256	256	6, 5
BS	SDK [7]	Blackscholes	2	480	128	8
CP	Sim [8]	Coulombic potential	1	1,024	128	8
FWT	SDK [7]	Fast walsh transform	13	1,472	315.08	Avg 4.62
KMS	Rodinia [10]	Kmeans	3	363	256	6
LIB	Sim [8]	LIBOR Monte Carlo	2	256	64	8
MC	SDK [7]	Monte Carlo reduction	2	384	192	8, 5
NN	Sim [8]	Neural network	4	432	120.75	Avg 7
SLA	SDK [7]	Scan of large arrays	6	387	256	6
SP	SDK [7]	Scalar product	1	128	256	6
ST	Parboil [11]	Stencil	1	128	128	8

#K Kernel count, *#C* CTA count, *CS* Average CTA size, *CpC* CTAs per core

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

4.2 Effectiveness

We first evaluate the performance of CLASO on the GPGPU baseline architecture i.e. the speedup upon LRR. We choose the tuning parameters ($p_A = 1$, $p_L = 0$) (CLASO10) and ($p_A = 2$, $p_L = 0$) (CLASO20) in this evaluation, because they perform the best speedups. The parameter sensitivity is discussed in Subsect. 4.4. Figure 11 illustrates the speedup of CLASO compared with LRR, where CLASO improves the performance of all benchmarks. The average speedup is 8.4 %, and the highest speedup comes up in CP and SP, which are 26.6 and 26.1 % respectively.

The CTA load balance and the TLP efficiency are illustrated in Fig. 12. CLASO does improve the CTA load balance, and reduces the idle cycles for all benchmarks by 52.4 % on average. Some benchmarks suffer from a serious load imbalance can be corrected with a significant drop of the idle cycles such as BS, CP, and SP. Because CLASO distributes all the CTAs among cores, the faster executing cores receive less CTAs in CLASO than in the baseline scheduler. Therefore, CLASO weakens the TLP efficiency as shown in Fig. 12b. Some benchmarks that favor high TLP such as BS and CP present much higher scoreboard conflicts in CLASO. However, CLASO tries

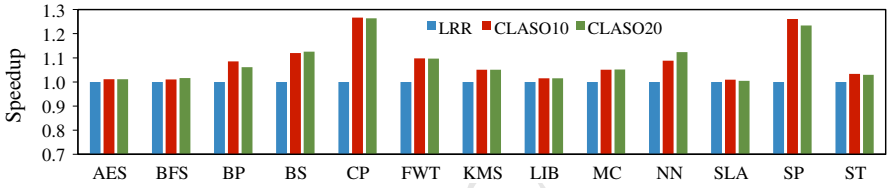

Fig. 11 The CLASO speedup when used with Loose Round Robin (LRR) scheduler. All the benchmarks achieve speedup that is 8.4 % on average with the highest of 26.6 % in CP

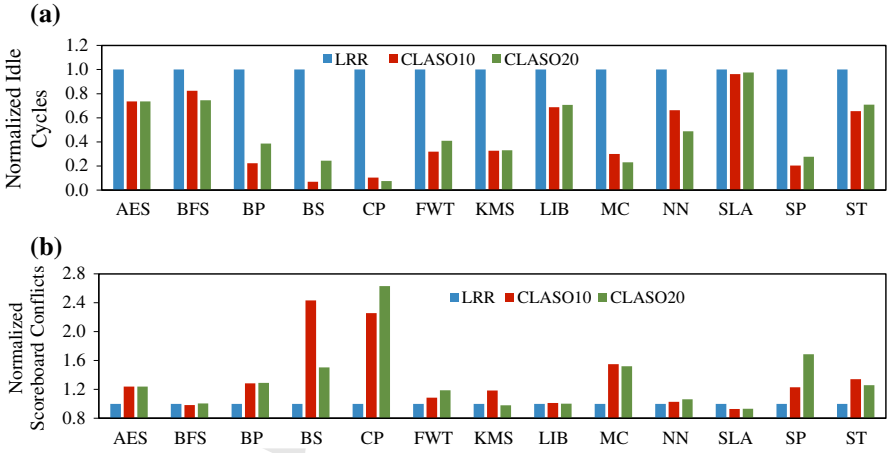

Fig. 12 The profiling result of CLASO upon LRR scheduler. CLASO reduce the idle cycles caused by CTA load imbalance by 52.4 % on average, which mainly contributes to the speedup, while CLASO increases the scoreboard conflicts and alleviates the TLP for most benchmarks. **a** Normalized idle cycles, **b** Normalized scoreboard conflicts

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

its best to keep TLP untouched, and only refuses the CTA issuance request when the credit exhausted.

The load imbalance and the TLP sensitivity are two important factors for the CLASO performance efficiency. CP achieves the highest performance because of the significant idle cycle decrease (92.5 %). The high TLP sensitivity of CP leads it achieving a similar speedup as SP whose idle cycles decrease by 80 %, but scoreboard conflicts increase by only 23 % in CLASO10. Benchmarks such as AES (1.1 %) and MC (5.1 %) do not achieves a high speedup, because the high TLP sensitivity evens up the benefit from the idle cycle decrease. SLA achieves a low speedup (1.0 %) because it does not exhibit a significant load imbalance. BS achieves a lower speedup (12.5 %) compared with CP, because not only BS exhibits an insignificant load imbalance, but also other factors such as the L1 locality affect the final speedup. Nevertheless, CLASO always improves the benchmarks' performance by improving the load balance.

4.3 Compatibility with Other Scheduling Policies

We test the effectiveness of CLASO when it is used with TL, OWL, and DYNCTA to evaluate its compatibility. Figure 13 illustrates the speedups upon these schedulers, while Figs. 14 and 15 show the normalized idle cycles and the normalized scoreboard conflicts respectively. CLASO achieves speedups in all benchmarks with 7.4 % upon TL, 4.4 % upon OWL, and 3.5 % upon DYNCTA on average, and thus it verifies that

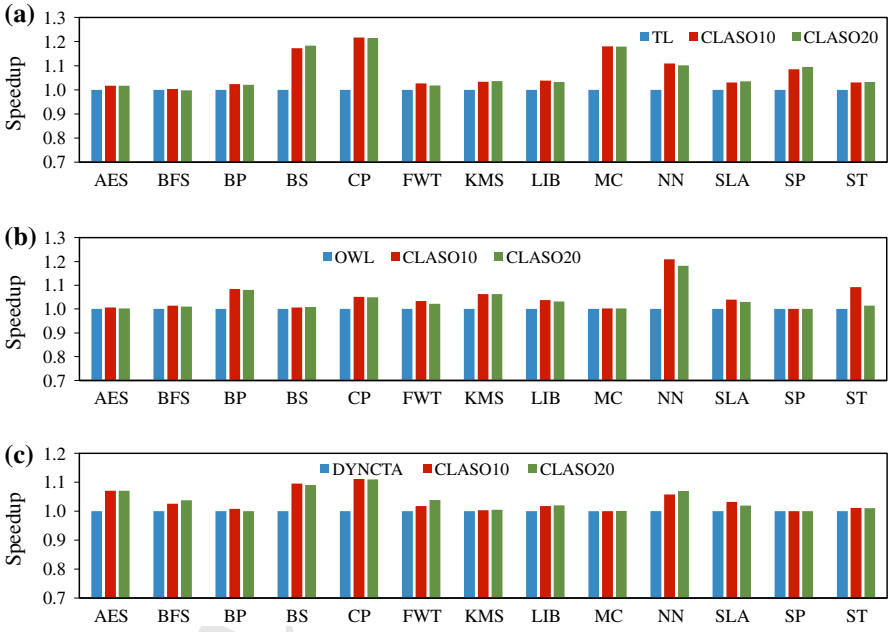

Fig. 13 The speedups of CLASO upon different warp and CTA schedulers. CLASO is compatible with these schedulers well, and achieves 5.1 % speedup on average where the highest is 18.3 %. **a** TL warp scheduler, **b** OWL warp scheduler, **c** DYNCTA CTA scheduler

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

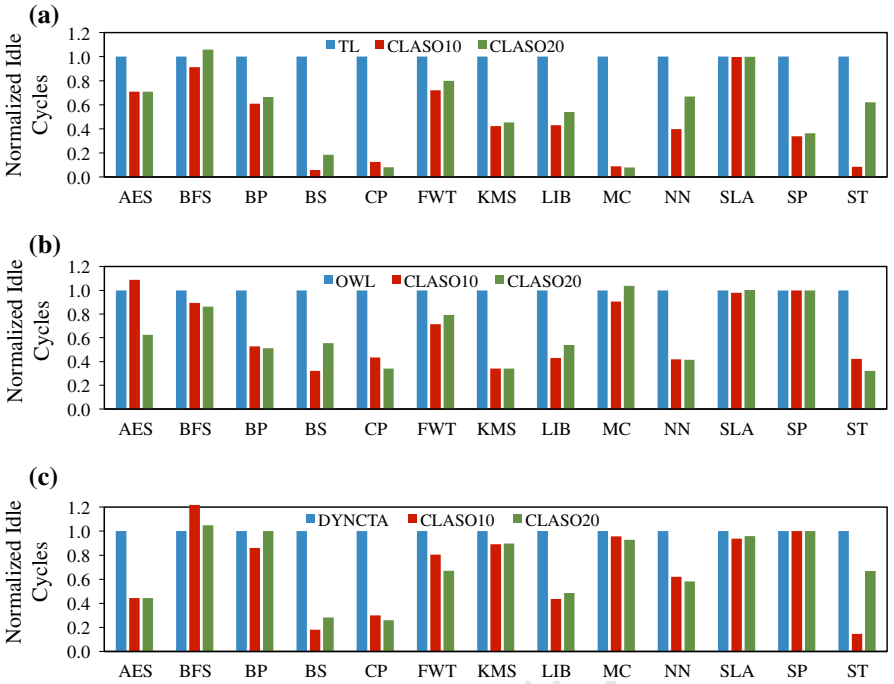

Fig. 14 The normalized idle cycles of CLASO upon different warp and CTA schedulers. CLASO effectively improves the CTA load balance for the underlying scheduler. **a** TL warp scheduler, **b** OWL warp scheduler, **c** DYNCTA CTA scheduler

CLASO is effectively compatible with existing warp and CTA schedulers. The highest speedup is 18.3 % that comes up in BS upon TL. TL, OWL, and DYNCTA do not take account of the load imbalance, while the fetch groups in TL, the warp preemption in OWL, and CTA pausing mechanism in DYNCTA diversify the execution time between CTAs inside a same core that weakens the load imbalance. It is the reason why CLASO still achieves speedups upon these schedulers but the speedups do not as high as upon LRR. Interestingly, some benchmarks such as MC upon TL (18.0 %) achieve higher speed up than upon LRR (5.1 %). Benchmarks like SP upon OWL and DYNCTA do not achieve any speedup because they have already achieve the load balance when using OWL and DYNCTA. It shows that CLASO helps improving the CTA load balance while maintaining the performance benefits from underlying scheduler.

CLASO reduces the idle cycles for most benchmarks upon TL, OWL, and DYNCTA. Because of the diversified CTA execution time, the tightest load balance (CLASO10) may not be the best choice in these schedulers. That is why AES upon OWL and BFS upon DYNCTA present higher idle cycles in CLASO. Increasing the tuning parameters to a looser CTA limit (CLASO20) is a better choice. For example, CLASO20 (3.7 %) achieves a higher speedup than CLASO10 (2.5 %) in BFS upon DYNCTA.

Except some TLP sensitive benchmarks such as BS, CP and ST, most benchmarks upon these three schedulers do not suffer from a significant scoreboard conflict increase

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

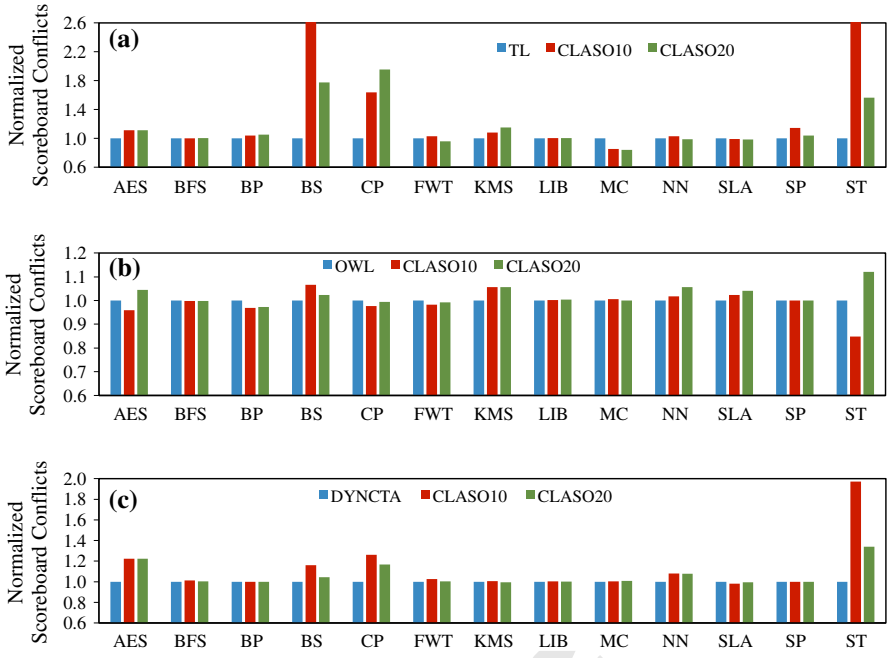

Fig. 15 The normalized scoreboard conflicts of CLASO upon different warp and CTA schedulers. **a** TL warp scheduler, **b** OWL warp scheduler, **c** DYNCTA CTA scheduler

by employing CLASO. TL, OWL, and DYNCTA exploit performance improvement mainly by improving TLP efficiency. CLASO maintains their effort as much as possible.

4.4 Parameter Sensitivity

We test the different tuning parameters to evaluate the sensitivity of CLASO. Figure 16 demonstrates the performance in different tuning parameters, where CLASO XY denotes ($p_A = X$, $p_L = Y$). We only demonstrate the results upon LRR because different underlying schedulers exhibit very similar tendencies while LRR exhibits the clearest tendency. There is no significant performance tendency between CLASO10

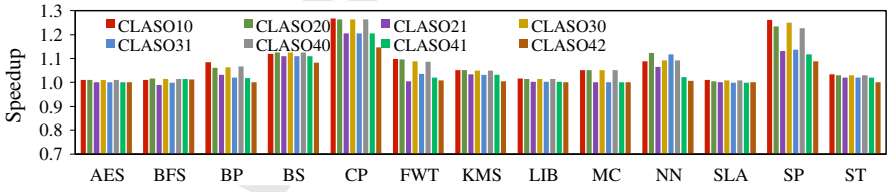

Fig. 16 The speedups of CLASO in different tuning parameters upon LRR. The performance decreases while loosing the limit on issuable CTAs by increasing the tuning parameters

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

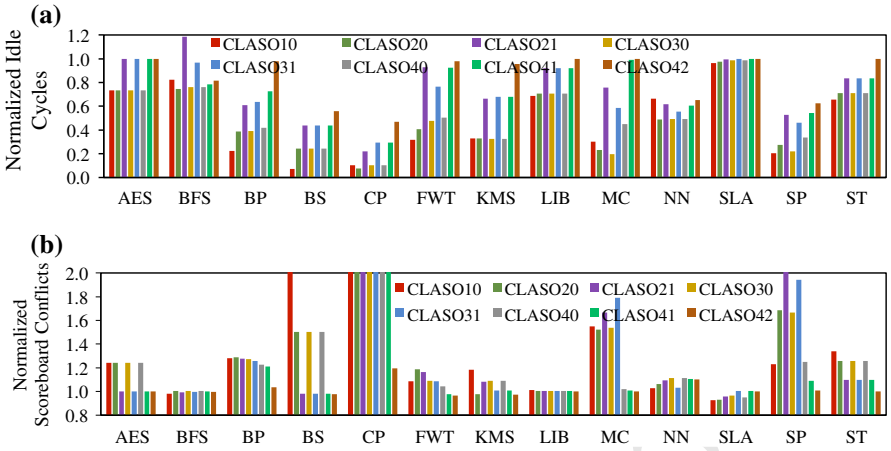

Fig. 17 The profiling results of CLASO in different tuning parameters upon LRR, where increasing tuning parameters usually leads to more idle cycles (i.e. increasing the load imbalance), but lower scoreboard conflicts (i.e. a higher TLP efficiency). **a** Normalized idle cycles, **b** Normalized scoreboard conflicts

and CLASO20, because CLASO10 gives an ideal scheduling decision for many benchmarks, but gives too tight limit to other benchmarks. The speedup gradually disappears while increasing p_A and p_L. It is why we recommend CLASO10 and CLA20 as the best choice. p_L is more sensitive than p_A, i.e. increasing p_L decreases the performance more significantly than increasing p_A. The performance of AES, LIB, MC, SLA, and ST in CLASO42 is the same as in the baseline scheduler, so CLASO degenerates to the baseline scheduler if the tuning parameters are excessively increased.

The profiling results are illustrated in Fig. 17. Because the idle cycles increase, the load balance efficiency gradually disappears while increasing the tuning parameters. Similar to the performance, p_L is more sensitive than p_A in idle cycles. CLASO10 and CLASO20 exhibit the best load balance in all the benchmarks. The load imbalance prefers distributing CTAs into a subset of GPU cores instead of all cores, which presents a higher TLP efficiency, so the scoreboard conflicts decrease while increasing the tuning parameters. Some exceptions such as MC and SP present higher scoreboard conflicts in larger tuning parameters because of some accidental CTA issuances to the late finished cores, where only one or two CTAs are issued on those cores and lead to high scoreboard conflicts.

5 Related Work

Most CTA scheduling policies were proposed for TLP efficiency. Kayiran et al. [3] proposed two CTA schedulers named DYNCTA. DYNCTA dynamically changes the number of concurrent CTAs on each core to achieve the best tradeoff between the TLP efficiency and the memory conflicts. Lee et al. [12] followed the same observation on TLP, and proposed Lazy CTA Scheduling and Block CTA Scheduling to tune the TLP. Adriaens et al. [13] highlighted the importance of spacial scheduling in GPGPU, and

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

discussed the efficiency of some CTA scheduling policies. In this work, we focus on the CTA load imbalance, and propose a CTA scheduling optimization as a patch to current scheduling system.

A warp scheduler targets to give an efficient executing order of warps. LRR is a simple and commonly used scheduling policy, but does not perform well in many cases. Narasiman et al. [1] proposed a two-level warp scheduler (TL). TL organizes warps into fetch groups. The lower level schedules the warps in a single fetch group in round robin, and when all these warps come to a long waiting operation, the higher level switches to the next fetch group. TL improves the resource utility by overlapping the data access and the computation. Jog et al. [14] proposed a variant of TL named prefetching-aware warp scheduling policy (PA) to alleviate the back-to-back caching and improve the memory bank-level parallelism. Jog et al. [2] proposed another upgrade to TL named cooperative thread array aware scheduling policy (OWL). The warp preemption mechanism is employed in OWL to improve the L1 cache hit rate. Gebhart et al. [15] optimized TL for the energy efficiency. Rogers et al. [16] proposed a cache-conscious wavefront scheduling (CCWS) to improve the L1 cache efficiency by dynamically tuning the number of active warps. Dynamic warp subdivision (DWS) proposed by Meng et al. [17], dynamic warp formation (DWF) proposed by Fung et al. [18], thread block compaction (TBC) proposed by Fung et al. [19], and simultaneous branch interweaving (SBI) and simultaneous warp interweaving (SWI) proposed by Brunie et al. [20] are control flow aware schedulers. All these schedulers did not take account of the load balance. We propose CLASO in this work that can be cooperate with these schedulers to improve their load balance and the performance.

The memory request scheduling policies are also important to GPGPU performance. Jia et al. [21] used memory request reordering and cache bypassing to improve the GPGPU per-thread cache capacity. Jog et al. [22] propose FR-RR-FCFS to improve a load balance between concurrent executing kernels. In this work we found the memory access load imbalance among GPU cores is a key factor to the CTA load imbalance. We give a CTA scheduling scheme instead of a memory scheduling policy to prevent a performance decrease from row locality violation.

The GPGPU load balance was also discussed in some other work from different aspects. Lakshminarayana et al. [23] studied and pointed out the importance of the fairness of instruction fetches and memory accesses to GPGPU performance. Chen et al. [24] propose a new programming model with fine-grain compute tasks to improve the load balance. CLASO improves the GPGPU load balance in scheduling system, so the application can enjoy the benefit without changing any source codes.

6 Conclusions

In this work, we investigate the CTA load balance problem in the current GPGPU schedulers. We find out the core-imbalanced memory accesses are the key factor of this issue. We propose a credit-based load-balance-aware scheduling optimization scheme (CLASO) as a patch piggybacked to existing GPGPU scheduling system to improve the CTA load balance. The basic idea of CLASO is to limit the amount of the issuable CTAs on each core by using credits. Local and global credits cooperate to

Journal: 10766-IJPP Article No.: 0318 TYPESET DISK LE CP Disp.:2014/8/9 Pages: 21 Layout: Small-X

avoid the drawback of the coarse-grain credit allocation. Two tuning parameters help improving the robustness to the diverse execution time between CTAs. We simulate CLASO using GPGPU-Sim, and conduct experiments to evaluate its effectiveness. The results show that CLASO effectively improves the CTA load balance by decreasing 52.4 % idle cycles on average. The speedup upon the baseline scheduler achieves up to 26.6 % in the benchmarks. CLASO is also compatible with other existing warp and CTA schedulers, and exhibits a speedup up to 18.3 % upon TL, OWL, and DYNCTA schedulers in our experiments.

Acknowledgments This research is partially sponsored by the U.S. National Science Foundation (NSF) Grants CCF-1102624 and CNS-1218960, and National Natural Science Foundation of China grants 61033012 and 11372067. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

1. Narasiman, V., Shebanow, M., Lee, C. et al.: Improving GPU performance via large warps and two-level warp scheduling. In: International Symposium on Microarchitecture, pp. 308–317 (2011)
2. Jog, A., Kayiran, O., Nachiappan, N. et al.: OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance. In: International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 395–406 (2013)
3. Kayiran, O., Jog, A., Kandermir, M. et al.: Neither more nor less: optimizing thread-level parallelism for GPGPUs. In: International Conference on Parallel Architectures and Compilation Techniques, pp. 157–166 (2013)
4. NVIDIA: CUDA C Programming Guide (2012) http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
5. Khronos Group: The open standard for parallel programming of heterogeneous systems (2013) http://www.khronos.org/opencl/
6. NVIDIA: NVIDIA Visual Profiler (2014) https://developer.nvidia.com/nvidia-visual-profiler
7. NVIDIA: CUDA C/C++ SDK code samples (2011) http://www.nvidia.com/cuda-cc-sdk-code-samples
8. Bakhoda, A., Yuan, G., Fung, W. et al.: Analyzing CUDA workloads using a detailed GPU simulator. In: International Symposium on Performance Analysis of Systems and Software, pp. 163–174 (2009)
9. NVIDIA: Tesla C2050 / C2070 GPU computing processor (2010). http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C2050_C2070_jul10_lores.pdf
10. Che, S., Boyer, M., Meng, J. et al.: Rodinia: a benchmark suite for heterogeneous computing. In: International Symposium on Workload Characterization, pp. 44–54 (2009)
11. Stratton, J.A., Rodrigues, C., Sung, I.J. et al.: Parboil: a revised benchmark suite for scientific and commercial throughput computing. Tech. Rep. IMPACT-12-01 University of Illinois at Urbana-Champaign (2012)
12. Lee, M., Song, S., Moon, J. et al.: Improving GPGPU resource utilization through alternative thread block scheduling. In: International Symposium on High Performance Computer Architecture, pp. 263–273 (2014)
13. Adriaens, J., Compton, K., Kim, N. et al.: The case for GPGPU spatial multitasking. In: International Symposium on High Performance Computer Architecture, pp. 1–12 (2012)
14. Jog, A., Kayiran, O., Mishra, A. et al.: Orchestrated scheduling and prefetching for GPGPUs. In: International Symposium on Computer Architecture, pp. 332–343 (2013)
15. Gebhart, M., Johnson, D.R., Tarjan, D. et al.: Energy-efficient mechanisms for managing thread context in throughput processors. In: International Symposium on Computer Architecture, pp. 235–246 (2011)
16. Rogers, T., O'Connor, M., Aamodt, T. et al.: Cache-conscious wavefront scheduling. In: International Symposium on Microarchitecture, pp. 72–83 (2012)
17. Meng, J., Tarjan, D., Skadron, K.: Dynamic warp subdivision for integrated branch and memory divergence tolerance. In: International Symposium on Computer Architecture, pp. 235–246 (2010)

18. Fung, W.W.L., Sham, I., Yuan, G. et al.: Dynamic warp formation and scheduling for efficient GPU control flow. In: International Symposium on Microarchitecture, pp. 407–420 (2007)
19. Fung, W., Aamodt, T.: Thread block compaction for efficient SIMT control flow. In: International Symposium on High Performance Computer Architecture, pp. 25–36 (2011)
20. Brunie, N., Collange, S., Diamos, G.: Thread block compaction for efficient SIMT control flow. In: International Symposium on Computer Architecture, pp. 49–60 (2012)
21. Jia, W., Shaw, K.A., Martonosi, M.: MRPB: memory request prioritization for massively parallel processors. In: International Symposium on High Performance Computer Architecture, pp. 274–285 (2014)
22. Jog, A., Bolotin, E., Guz, Z. et al.: Application-aware memory system for fair and efficient execution of concurrent GPGPU applications. In: Workshop on General Purpose Processing Using GPUs, pp. 1–8 (2014)
23. Lakshminarayana, N.B., Kim, H.: Effect of instruction fetch and memory scheduling on GPU performance. In: Workshop on Language, Compiler, and Architecture Support for GPGPU, pp. 1–10 (2010)
24. Chen, L., Villa, O., Krishnamoorthy, S. et al.: Dynamic load balancing on single- and multi-GPU systems. In: IEEE International Symposium on Parallel and Distributed Processing, pp. 1–12 (2010)

Journal: **10766-IJPP** Article No.: **0318** TYPESET DISK LE CP Disp.:**2014/8/9** Pages: **21** Layout: **Small-X**

Journal: 10766
Article: 318

Springer
the language of science

Author Proof

Author Query Form

Please ensure you fill out your response to the queries raised below and return this form along with your corrections

Dear Author

During the process of typesetting your article, the following queries have arisen. Please check your typeset proof carefully against the queries listed below and mark the necessary changes either directly on the proof/online grid or in the 'Author's response' area provided below

Query	Details required	Author's response
1.	Please check and confirm that the authors and their respective affiliations have been correctly identified and amend if necessary.	
2.	As per the information provided by the publisher, Fig. 8 will be black and white in print; hence, please confirm whether we can add "colour figure online" to the caption.	