Implementation of CAPRI (Compaction Adequacy Predictor) for Handling Branch Divergence in GPGPU Simulator

Dhruv Sheth
Computer Engineering Department
North Carolina State University
Raleigh, USA
dsheth@ncsu.edu

Prit Gala
Computer Engineering Department
North Carolina State University
Raleigh, USA
pgala@ncsu.edu

Veerakumar Natarajan
Computer Engineering Department
North Carolina State University
Raleigh, USA
vnatara4@ncsu.edu

Kunal Ochani
Computer Engineering Department
North Carolina State University
Raleigh, USA
kdochani@ncsu.edu

Abstract

Control flow divergence is handled in GPGPUs by masked thread execution. Thus, when general purpose workloads are executed which involves a lot of irregular control flow we use masking. Whenever we encounter a branch instruction there is serialization of code execution and depending upon the mask, the output of the SIMD lane is used or discarded. There is a performance degradation because the issue slots of the masked lanes are wasted. This degradation can be mitigated using dynamic thread block compaction (TBC) which consists of unmasked threads. However, in certain workloads TBC adds some overhead instead of performance improvement. This paper proposes a new technique, CAPRI, that eliminates this overhead by dynamically deciding the effectiveness and performs thread block compaction one when it is beneficial. CAPRI is a simple branch predictor inspired structure which has a predictor accuracy of 99.8% and 86.6% for non-divergent and divergent workloads respectively. CAPRI always outperforms its previous techniques of WPDOM which never performs compaction and Thread Block Compaction which always stalls the warps on all divergent branches.

Introduction

To achieve high performance in terms of both power and cost, GPUs utilize single instruction multiple-data (SIMD) based execution. SIMD improves performance by partitioning the register storage and by exhibiting data level parallelism as task level parallelism. However, SIMD handles the control instructions by applying masked execution. This masked execution introduces serialization which often degrades the performance. Hence, control divergence should be given careful consideration in order to mitigate the performance degradation.

GPU's handle control divergence using either SIMT stack (Post Dominator), Predication or Predication with unanimous instruction approach. SIMT stack approach requires the use of active mask. Active mask is used to discard the output of the threads are not supposed to execute in the current divergent path. However, this type of technique results in low SIMD utilization which becomes worse with increasing number of SIMD execution lanes.

Several techniques have been introduced to improve this SIMD utilization for control divergence. Thread Block Compaction(TBC) is one such technique that dynamically forms new warps within a thread block with the aim to reduce serialization when the warps execute branch instruction. This technique has an essential requirement of stalling the warps in a thread block to consider them for compaction when the branch instruction is detected. If the number of warps before and after compaction is the same, this negate the benefit of warp compaction and becomes an overhead. This issue is addressed in this project.

The objective of the project is to introduce a hardware structure that stalls only those warps that have a high likelihood of benefiting from compaction while allowing other warps to bypass compaction and continue execution. Compaction occurs only at selected branches, while for other branches warps can continue executing in regular manner without compaction. This technique is known as CAPRI (Compaction Adequacy Predictor). It comprises a decision logic and prediction hardware along with a compaction unit in order to determine the compaction effectiveness. TBC and CAPRI are discussed in the sections below. In this project, TBC and CAPRI are implemented in GPGPU simulator.

Background & Related Work

1. Post Dominator Algorithm

Post Dominator is the most basic approach used in GPUs for handling control divergence. This approach utilizes the active mask of each warps which are stored in reconvergence stack (SIMT stack). A separate reconvergence stack is maintained per warp. When a divergent branch is detected, true and false path must be executed one after the another which partially serializes the execution. Each thread executes both the paths but the results of the threads with non-active masks are discarded.

SIMT stack consists of reconvergence PC of the branch which can be computed at compile time and is also known as Post Dominator of that branch. When a branch is encountered, the active mask for the branch and reconvergent PC are pushed into the stack. In addition, if there is divergence, the corresponding entries for the taken and not-taken paths along with their next PC are pushed into the stack. Now, instructions are fetched from the PC in the next PC entry of the taken path. When all the instructions of the taken path are executed, the PC matches with the reconvergence PC at the TOS and this entry is popped out of the stack. Now, the hardware starts executing the not-taken path.

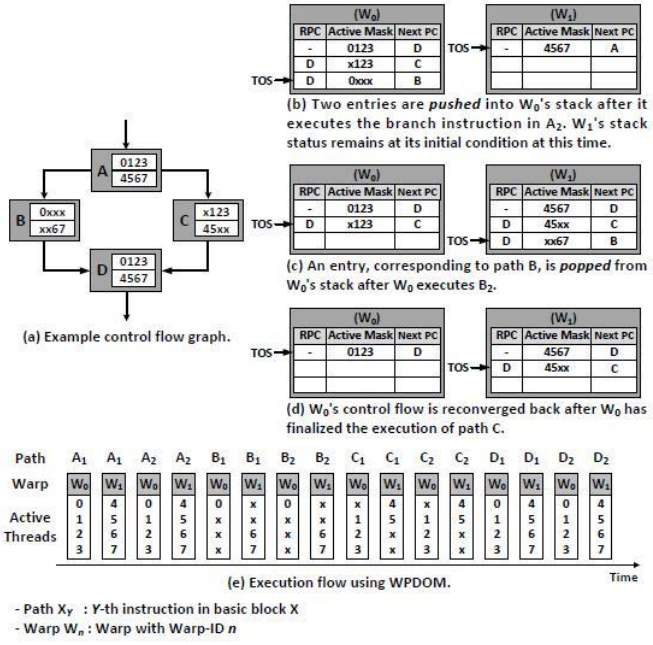


Fig. 1. Example showing WPDOM handling branch divergence

However, this method has a drawback when there are nested branches, the number of active threads in each path of the branch decrease which in turn reduces the SIMD utilization.

2. Thread Block Compaction.

WPDOM suffers from SIMD utilization due to serial execution during branch divergence. This SIMD utilization can be improved by performing dynamic thread block compaction of all active threads of a particular branch into minimum number of warps. Instead of having a single SIMT stack per warp a new thread block wide SIMT stack is used. This single stack is used to determine the point when the active masks of the true and false paths are fully known in order to apply maximal compaction. This stack structure tends to introduce a barrier after each branch so that the compaction unit has access to entire active mask of the CTA. Both branching and reconvergence can only occur after all active warps have synchronized.

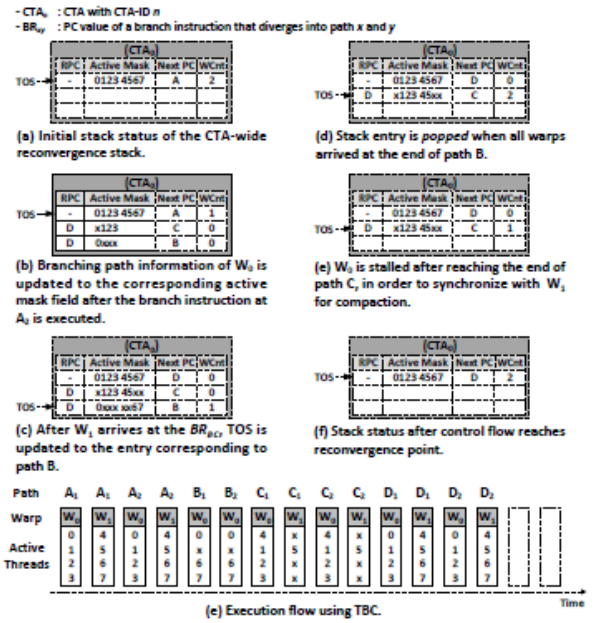


Fig. 2 High level operation of TBC

A new field, Wcnt, is added in the SIMT stack which has a counter for determining the number of active warps in the current control follow path. Wcnt indicates the number of warps that the branch instruction should wait for. Each time a warp executes a branch instruction or reaches reconvergence point, Wcnt of the TOS is decremented. When all active warps have either reach the end of the basic block and executed the branch instruction or reach the reconvergence point

Wcnt becomes 0 and next basic block can start its execution.

When Wcnt becomes 0, the warp compaction unit (WCU) consisting of a set of priority encoders receives the CTA wide active mask to perform warp compaction while maintaining the fixed association of each thread to its SIMD lane. The WCU unit generates the new active mask and the number of warps required for execution.

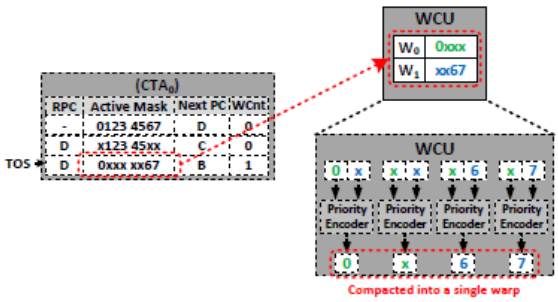


Fig. 3 Warp Compaction Unit

In many workloads, compaction does not result into reduction in the number of warps as result of which a lot power is wasted to perform the compaction of ineffective branches and introduces unnecessary synchronization points.

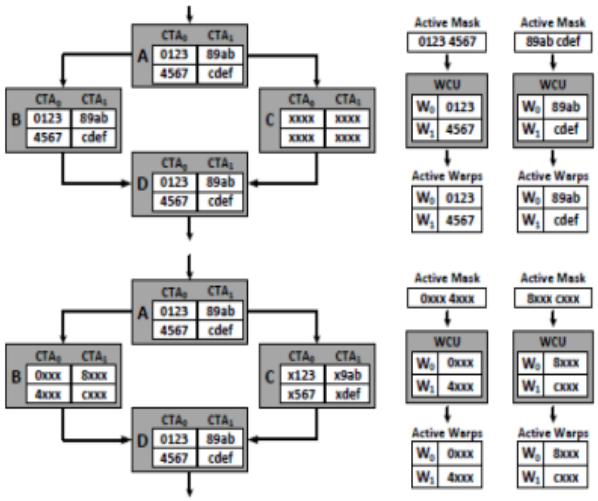


Fig. 4 Compaction in-effective branches

3. CAPRI

The idea of compacting the warps when there is divergent is not always effective. This is because a divergent branch may diverge in a similar way across all warps which makes compaction ineffective. As a result, the number of warps before and after compaction is the same. Similar is the case when there is a loop end branch which is non-divergent. When compaction is performed for the loop end branch, the number of warps after compaction is the same. This leads to drawbacks in terms of wastage of power in TBC, synchronization overheads and in some cases memory divergence.

To overcome this issue, a hardware structure known as Compaction Adequacy Predictor (CAPRI) is used. The objective of the CAPRI is to predict whether it would be beneficial to stall the warp for compaction once the warp arrives at branch instruction. CAPRI consists of three main components:

1. **Compaction Adequacy Prediction Table (CAPT)** which keeps track of adequacy history.
2. **Decision Logic** which decides whether to stall the warp for compaction.
3. **Warp Compaction Unit (WCU)** which is used to perform compaction and determines the compaction adequacy.

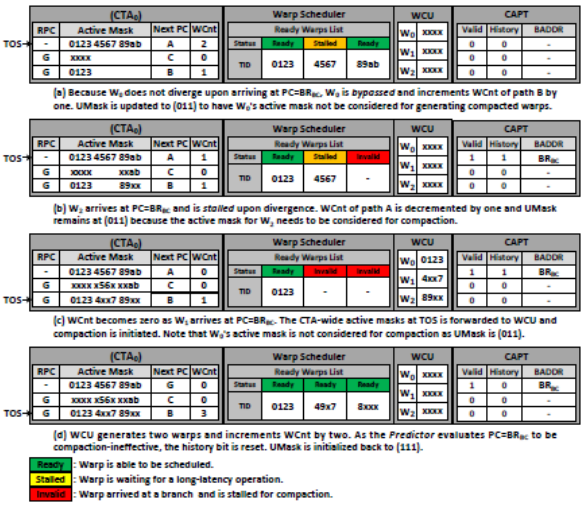


Fig. 5 Different components in CAPRI

In addition, CAPRI also contains one bit per warp known as Update Mask (UMASK) which is used to indicate whether a warp should be taken for compaction. UMASK bit of that warp is reset.

Each CAPT entry consist of three fields: Valid bit which is used to indicate whether the entry is valid or not, History bit to predict if the compaction would be effective & BADDR to store the PC of a particular branch instruction.

When a warp arrives at branch there are three possible outcomes. As shown in Fig. 5, since warp 0 is non-divergent at $BR_{B\text{-}C}$, there is not benefit in stalling this warp for compaction. Hence, CAPRI bypasses this warp from the CAPT table and WCU.

Warp 2 is divergent, but there is no entry in CAPT for branch BR_{B-C}. In this case, the warp is conservatively considered as compaction effective. An entry is created for that branch in CAPT and history bit is set. When warp 2 arrive at BR_{B-C}, the branch entry is present in the CAPT. In this case, history bit is used to decide if the warp should be considered for compaction.

Warp Compaction Unit (WCU) performs compaction based on the UMASK bits of the warp. Based on the number of warps formed after the compaction, the WCnt entry in the stack is incremented. To find the compaction effectiveness, WCU performs the prediction by compacting all the warps in a thread block regardless of the UMASK bits. This is necessary because warps which did not participate in compaction might have benefitted from compaction. If the number of warps formed after this compaction is equal to number of warps before compaction, then history bit in the CAPT is reset.

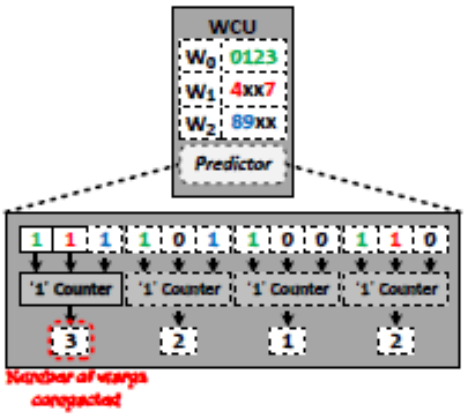


Fig 6. Evaluating compaction-adequacy

Methodology

CAPRI is implemented in GPGPU-Sim version 3.2.2 which is a detailed simulation model of contemporary GPUs. The GPGPU-sim is modified to implement TBC and CAPRI. Prediction logic used for CAPRI is 1-bit history table which holds latest configuration of a branch instruction. Initially entire program is allowed to run on the simulator and instruction trace is collected which is processed to search for divergent instructions. Each instruction is stored along with thread block ID, warp ID, PC and active mask associated with it. In case of divergent instructions, we are considering active mask of all the warps in that particular thread block executing the same instruction.

These divergent instructions are processed using Warp Compaction Unit (WCU). WCU counts the number of warp needed for all the 32 lanes. This count gives the actual number of warp needed to execute that particular lane. Maximum value among column sum of 32 lanes gives the number of warps needed after compaction. For TBC,compaction is performed for every branch instruction. The number of warps saved after every branch is counted as tbc_warp_saved by subtracting the number of warps before compaction and after compaction.

For CAPRI, we maintain CAPT with following fields: PC, history bit. CAPT is indexed using PC of branch instruction. First time when a branch is encountered, compaction is performed to find minimum number of warps needed and then capri_warp_saved is updated with number of warps saved. If number of warps can be reduced from initial number of warps, then the history bit of corresponding field in CAPT is set true. Next time when the same branch instruction is encountered, compaction is performed only if history bit is true. If the history bit is false, the warp is allowed to bypass the branch. After every branch instruction, prediction is performed to check if the branch instruction is compaction adequate or not and CAPT is updated according to the results of the prediction. To handle nested branch instructions, a stack for every thread block is maintained. Increase in the SIMD utilization rate is calculated using number of divergent instructions, tbc_warp_saved and capri_warp_saved.

Results

```
__global__ void
vectorAdd(const float *A, const float *B, float *C, int numElements)
{
    int i = threadIdx.x;

    if ((i < 16) || (i >= 48 && i <  64) || (i >= 64 && i < 80) || (i >= 112 && i < 128))
    {
        C[i] = A[i] + B[i];
    } else {
            C[i] = A[i] - B[i];

        }
        if(i % 2 == 0)
            C[i] = C[i] * A[i];
        else
            C[i] = C[i] / B[i];
}
```

Fig. 7 Test kernel

In the above figure a code snippet of a test divergent kernel has been shown. Experiments have been conducted using various benchmarks and this test kernel. The results of these experiments are shown in the graphs below.

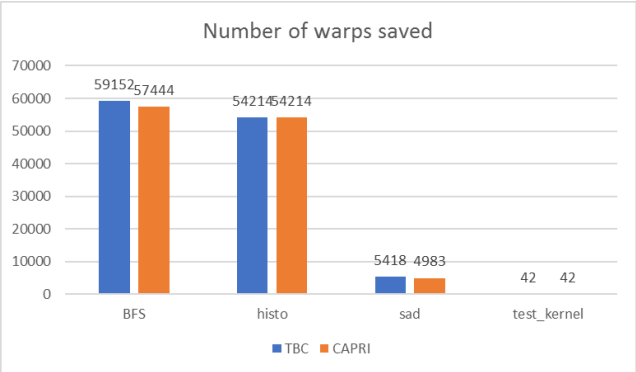


Fig.8 Number of warps saved

Compaction helps to reduce the number of warps after divergence. In the above figure, BFS is a divergent kernel and we observe a significant reduction in the number of warps using TBC and CAPRI. The test kernel being a small kernel has smaller number of warps issued and hence smaller warp savings.

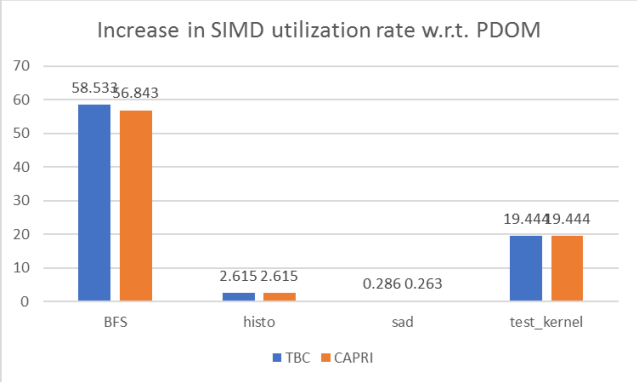
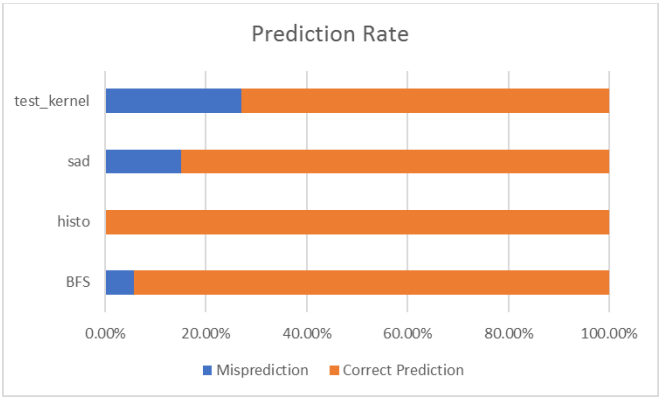


Fig. 9 Increase in SIMD utilization rate

Increase in the SIMD utilization is the key metric to measure the performance improvement. Since BFS is a divergent kernel, we observe a significant increase in the SIMD utilization. TBC performs compaction at every branch instruction while CAPRI achieves around 2% close performance to TBC. CAPRI gives lesser performance due to the mispredictions from the CAPT table. For the remaining benchmarks, we observe that CAPRI gives almost same SIMD utilization rate as TBC who together perform better than the baseline PDOM technique.

Since CAPRI uses prediction method to determine whether or not to perform warp compaction, this method involves some inaccuracies in terms of mispredictions. We have used the 1-bit counter for prediction. The lesser is the misprediction rate, closer will be the SIMD utilization rate of CAPRI and TBC. This trend can be observed from the above graphs. In case of the histo benchmark, we have 0% misprediction rate and hence the SIMD utilization rate of CAPRI and TBC is the same. The gap in the SIMD utilization rate between CAPRI and TBC for BFS is due the mispredictions in the decisions to perform warp compaction.



Conclusion

In this paper a novel idea about compaction adequacy predictor has been discussed which overcomes the fundamental deficiency of decreased performance from unnecessary synchronization. Majority of the branches are not benefitted from compaction as either the warps do not diverge much or they diverge in the same pattern which renders them compaction ineffective. This overhead of unnecessary synchronization is eliminated in CAPRI by performing warp compaction only when the adequacy prediction is true. When the adequacy prediction is false warp compaction is bypassed. Thus, CAPRI gives improved performance whenever there is room for improvement.

References

[1] Minsoo Rhu, and Mattan Erez. CAPRI: Prediction of Compaction-Adequacy for Handling Control-Divergence in GPGPU Architectures. In Proceedings of the International Symposium on Computer Architecture (ISCA'12). June, 2012.

[2] W. W. Fung and T. M. Aamodt. Thread Block Compaction for Efficient SIMT Control Flow. In 17th International Symposium on High Performance Computer Architecture (HPCA-17), February 2011.

[3] GPGPU- Sim: https://www.gpgpu-sim.org.