Dynamic SIMD Re-convergence with Paired-Path Comparison

Yun-Chi Huang[†], Kuan-Chieh Hsu[†], Wan-shan Hsieh[†], Chen-Chieh Wang, Chia-Han Lu, and Chung-Ho Chen[†]

[†]Inst. of Computer and Communication Engineering, National Cheng-Kung University, Tainan, Taiwan

Industrial Technology Research Institute, Hsinchu, Taiwan

Abstract—**SIMD divergence is one of the critical factors that decrease the hardware utilization in contemporary GPGPUs (General Purpose Graphic Processor Unit). Both the re-convergence scheme and control flow detection have to be well considered. In the emerging HSA (Heterogeneous System Architecture) platform, we develop an effective dynamic stack-based re-convergence scheme that can be implemented without the insertion of re-convergence instructions generated by the finalizer. The stack keeps track of the minimal necessary information of the taken and non- taken paths; the additional end-of-branch instruction insertion is no longer required under our design. Using the scheme we propose, the divergent warp dynamically re-converges at opportunistic re-convergence points. The activity factor improves for 13.36% on average from opportunistic early re- convergence in the unstructured control flow. Our design has eased the development of a finalizer that no longer needs to reason about the re-convergence point after a branch divergence, especially for unstructured control flow.**

Keywords— Branch divergence, GPGPU, HSA, SIMD, warp

I. Introduction

Being a parallel computing device, GPU not only serves as a graphic computing engine but recently is employed for general parallel applications. The distinctive SIMD (Single Instruction Multiple Data) feature of GPU architecture brings out the potential of high performance computing; yet the divergent control flow degrades the utilization of the parallel hardware resource.

In SIMD scheme, there are dozens of execution lanes performing the same instruction on different data. When a conditional branch instruction is executed, some of the active lanes will follow different control flow than others do. Since the divergent paths cannot be executed simultaneously due to SIMD design, both of them will be visited. When one of the divergent control paths is executed, the condition mask indicates which lanes are active during this path; whereas the others remain idle.

One of the common way handling control divergence is stack-based re-convergence scheme, such as the design used in contemporary GPUs [2][7]. The present stack-based re-convergence schemes all re-converge at the immediate post-dominator (IPDOM) points. The IPDOM is the point where all of the divergent paths leaving from the same conditional branch firstly re-converge [1]. Based on our observation, there are opportunistic early re-convergence points from which the IDPOM re-convergence can't benefit in an unstructured control flow kernel program. We propose a detection algorithm with dual-path stack structure to effectively support the opportunistic early re-convergence.

In the following, Section 2 introduces the background of warp-based GPU models and the categories of re-convergence schemes. Section 3 points out the observation and combination of previous work and our innovation. Section 4 provides the evaluation of the proposed mechanism. We conclude this paper in Section 5.

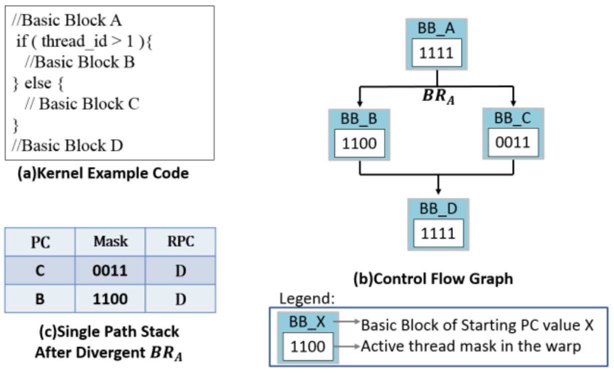
II. Backgroud

Conventional GPU contains multiple parallel computing units called SMs ("streaming multiprocessors" in Nvidia's terminology, and "SIMD units" in AMD's). The grid, a conceptual computation domain of a kernel, is defined in runtime stage, and each of the workgroups splitted from the grid is assigned to a corresponding streaming multiprocessor (SM) by a workgroup distribute unit. Also, the workitems in a workgroup may be divided into multiple warps due to SM hardware resource constraint. All workitems execute the same kernel program, but different workitems may execute in different control flow. As an example, a warp comprises 32 workitems in NVIDIA Fermi GPU architecture [10], and 64 workitems in AMD Evergreen Family Architecture [11]. The warp scheduler in the SM selects a ready warp, and fetches an instruction of the warp from the instruction memory. Each workitem is mapped to a processing lane in a SM, and all processing lanes execute the same instruction. When executing a warp, the processing lanes fetch data from per workitem's register file and execute the same instruction for active workitems. After executing an instruction, the results of a warp are written back to respective register file. A workgroup is finished if all warps of the workgroup are completed.

A. Stack-Based Re-convergence Schemes

Each workitem in a warp executes the same kernel code but may have independent control flow. If the workitems in a warp execute the same conditional branch instruction but yield different results, branch divergence or SIMD control divergence would occur. The commonly used mechanism for maintaining branch divergence is through divergence stacks [1][8].

Fig. 1 shows a code example of divergent branch BR_A and the corresponding control flow graph with stack status. As a warp encounters a divergent branch, the branch unit pushes 2 divergent stack entries that record the starting PC (Program Counter) values of the two following basic blocks (B and C) and their respective execution masks of both the warp-splits. The warp goes through either A-B-C-D or A-C-B-D flow to resolve branch divergence by referring to the stack entries.



```
//Basic Block A
 if ( thread_id > 1 ){
   //Basic Block B
} else {
   // Basic Block C
}
//Basic Block D
```

PC	Mask	RPC
C	0011	D
B	1100	D

(a)Kernel Example Code
BB_A
1111
BR_A
BB_B
1100
BB_C
0011
BB_D
1111
(b)Control Flow Graph
(c)Single Path Stack After Divergent BR_A
Legend:
BB_X
Basic Block of Starting PC value X
1100
Active thread mask in the warp

Fig. 1. Single path divergent stack status after encountering a divergent branch BR_A.

B. *Per-thread PCs Re-convergence Schemes*

Instead of using a shared stack that contains starting PCs in a warp, some GPUs employ per-thread per PCs (PTPCs) scheme to record status of workitems in a warp [1]. Using PTPCs, each workitem has a PC to track the execution flow. Employing PTPCs re-convergence has simpler control logic but higher hardware overhead. Besides, most of the comparisons of PCs are redundant when the control divergent case is rare.

C. *Re-convergence Detecting Methods*

The re-convergence detecting methods can be classified into *explicit* and *implicit* methods [1]. Explicit methods use compiler-generated instructions to indicate the re-convergence point through a finalizer. Specifically, the finalizer in HSA runtime system, for example, traverses the kernel code and intelligently inserts the end branch instruction at the IPDOM points. Implicit methods only rely on hardware re-convergence PC (RPC) detections without explicit instructions.

III. PAIRED-PATH COMPARISON RE-CONVERGENCE

We propose a paired-path comparison (PPC) algorithm that contains dual-path divergence stack to solve PC-based re-convergence. The dual-path stack (DPS) re-convergence scheme [2] is an alternative stack-based re-convergence scheme which records the information of two divergence warp-splits generated by a divergent branch on a stack entry. By recording information of both paths, DPS allows interleaved executions of warps in the SM and more memory access latency hiding opportunities.

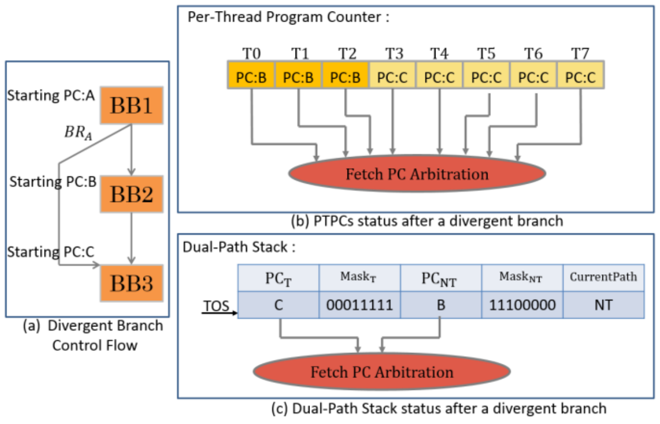
The PC-based re-convergence scheme detects warp re-convergence by comparing warp-splits PC values instead of using re-convergence instructions to indicate the re-convergence address, and this scheme may be benefited from opportunistic early re-convergence points in the unstructural control flow [4]. The control flow graphs of unstructural control flow can't be presented in hammock graphs [9] since some branches in unstructural control flow jump out of its If-else basic block, where the early re-convergence opportunities in the kernel may exist. This complicates the reasoning of the re-convergence point for the finalizer.

Our proposed PPC re-convergence scheme which can be used in both *explicit* and *implicit* control convergence method integrates the DPS with PC-based re-convergence. With PC-based re-convergence feature, the stack-based scheme with PPC is able to dynamically support re-convergence and opportunistic early re-convergence [4]. In the implicit control convergence scheme, the finalizer is no longer required to compute the re-convergence PC. The divergent warp can re-converge without any compiler hint during execution. This yields to the simplification of finalizer design.

A. *Observation*

Previous implicit stack-less PC-based re-convergence scheme uses PTPCs to store the different program counter values, as shown in Fig. 2.(a). If a warp consists of 8 threads, the implicit stack-less re-convergence scheme will require 8 PTPCs to record the divergent PC values of all threads, and it requires eight comparison modules between warp common PC and PTPCs. However, only two distinct program counter values in a warp are generated by a divergent branch. Numerous PTPCs share the same PC value, but they still need to be redundantly compared with the warp common PC separately.

The key observation is that only two divergent execution flows are generated from a divergent branch. No matter how many threads are assigned in a divergent execution flow, all PCs of the threads in the same execution flow share the same value. Two PCs on a dual-path stack entry are sufficient for tracking two divergent control flows. Fig. 3(c) presents the status of DPS after a divergent branch.



T0	T1	T2	T3	T4	T5	T6	T7
PC:B	PC:B	PC:B	PC:C	PC:C	PC:C	PC:C	PC:C

	PC_T	$Mask_T$	PC_{NT}	$Mask_{NT}$	CurrentPath
TOS	C	00011111	B	11100000	NT

Starting PC:A
BB1
BR_A
Starting PC:B
BB2
Starting PC:C
BB3
(a) Divergent Branch Control Flow
Per-Thread Program Counter :
Fetch PC Arbitration
(b) PTPCs status after a divergent branch
Dual-Path Stack :
Fetch PC Arbitration
(c) Dual-Path Stack status after a divergent branch

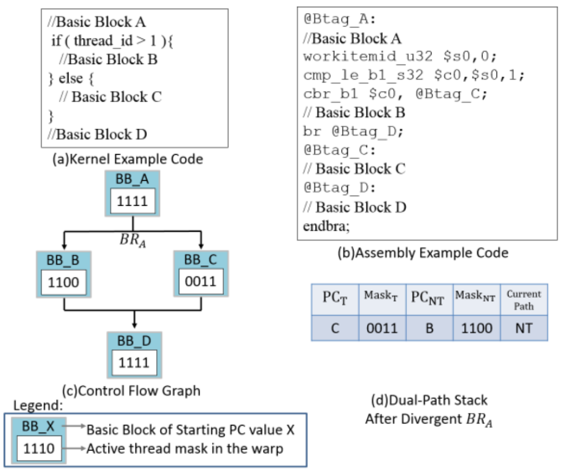
Fig. 2. PC arbitration of different re-convergence schemes

B. *Dynamic Re-convergence with PPC*

To support early re-convergence with divergence stack, previous researches need to construct a table implemented by content addressable memory (CAM) which is area costly to record warp splits' information during execution.

Instead of using a warp-split table to support opportunistic early re-convergence, our proposed dynamic paired-path comparison algorithm provides opportunity of early re-convergence with a dual-path stack which can be implemented effectively and benefits from the way of using stack pointer for searching and comparison at the top of the stack.

C. PC Arbitration With PPC



```
//Basic Block A
if ( thread_id > 1 ){
  //Basic Block B
} else {
  // Basic Block C
}
//Basic Block D
```

(a)Kernel Example Code

```
@Btag_A:
//Basic Block A
workitemid_u32 $s0,0;
cmp_le_b1_s32 $c0,$s0,1;
cbr_b1 $c0, @Btag_C;
// Basic Block B
br @Btag_D;
@Btag_C:
// Basic Block C
@Btag_D:
// Basic Block D
endbra;
```

(b)Assembly Example Code

PC_T	$Mask_T$	PC_{NT}	$Mask_{NT}$	Current Path
C	0011	B	1100	NT

(d)Dual-Path Stack After Divergent BR_A

Fig. 3. Dual path divergent stack status after encountering a divergent branch

The operation of paired-path comparison (PPC) for the detection of warp re-convergence is shown in Fig. 4. The next PC algorithm is shown in Fig. 5. When a divergent branch is encountered, the two divergent paths are pushed onto the stack (line 4, Fig. 5), then PPC function is invoked (line 31, Fig.5). The fetch unit compares the PC values of the taken path and the not-taken path. In the beginning, the two PC values are unequal, the current execution path starts with the path of smaller PC (line 10, Fig.4). The PC of the executed path is updated on the stack entry for non-divergent instructions (line 6 to 16, line 19 to 29, Fig. 5). If the two PC values equal to each other, the top of the stack (TOS) entry is deleted and the fetch unit continues execution on the parent level of divergent path for non-empty stack.

The difference between *implicit* and *explicit* schemes is how frequently the paired-path comparison must be performed during execution. The explicit scheme takes the paired-path comparison (PPC) when the warp encounters a re-convergence instruction while the implicit scheme performs the PPC for every instruction execution.

```
function PPCReconvergence()
1.      if !isDivergenceStackEmpty()
2.          if TOS.TakenPathPC == TOS.NotTakenPathPC
3.              DeleteTheTOSEntry()
4.              if TOS.CurrentExecutionPath == TakenPath
5.                  TOS.TakenPathPC ← CurrentWarpPC
6.              else
7.                  TOS.NotTakenPathPC ← CurrentWarpPC
8.              end if
9.          else
10.             if TOS.TakenPathPC < TOS.NotTakenPathPC
11.                 TOS.CurrentExecutionPath ← TakenPath
12.                 CurrentWarpPC ← TOS.TakenPathPC
13.             else
14.                 TOS.CurrentExecutionPath ← NotTakenPath
15.                 CurrentWarpPC ← TOS.NotTakenPathPC
16.             end if
17.         end if
18.     end if
```

Fig. 4. Detection of warp re-convergence.

```
Initial : stack set to empty
Input : Current_warpPC
Output : Next_warpPC
1.    while !KernelFinished() do
2.        if isBranchInstruction()
3.            if isDivergentBranch()
4.                push a new DPS entry
5.            else
6.                if isDivergenceStackEmpty()
7.                    CurrentWarpPC += BranchOffset + 8
8.                else
9.                    if TOS.CurrentExecutionPath == TakenPath
10.                       TOS.TakenPathPC += BranchOffset + 8
11.                       CurrentWarpPC ← TOS.TakenPathPC
12.                   else
13.                       TOS.NotTakenPathPC += BranchOffset + 8
14.                       CurrentWarpPC ← TOS.NotTakenPathPC
15.                   end if
16.               end if
17.           end if
18.       else
19.           if isDivergenceStackEmpty()
20.               CurrentWarpPC ← CurrentWarpPC + 8
21.           else
22.               if TOS.CurrentExecutionPath == TakenPath
23.                   TOS.TakenPathPC += 8
24.                   CurrentWarpPC ← TOS.TakenPathPC
25.               else
26.                   TOS.NotTakenPathPC += 8
27.                   CurrentWarpPC ← TOS.NotTakenPathPC
28.               end if
29.           end if
30.       end if
31.       PPCReconvergence()
32.       if isDivergenceStackEmpty()
33.           Next_warpPC ← CurrentWarpPC
34.       else
35.           if TOS.CurrentExecutionPath == TakenPath
36.               Next_warpPC ← TOS.TakenPathPC
37.           else
38.               Next_warpPC ← TOS.NotTakenPathPC
39.           end if
40.       end if
41.   end while
```

Fig. 5. Next PC algorithm: implicit paired-path comparison scheme with DPS.

IV. Evaluation

We model the proposed mechanism in a cycle-accurate HSAIL(Heterogeneous System Architecture Intermediate Language)-based[12] GPU simulation platform we develop, including a finalizer that translates the HSAIL code to our custom GPU ISA. The wave-front size of the streaming multiprocessor is 16, i.e., warp size. Several OpenCL structural control flow benchmarks from AMD SDK [5] and Rodinia [6] that have divergent branches, and three micro-benchmarks we write that have unstructural control flow are evaluated.

Activity factor is an SIMD efficiency measurement defined by Kerr et. al. [7]. The definition is the average ratio of masked threads in a warp to all active threads in the warp during dynamic instruction execution. The measurement shows the impact of SIMD divergence on the hardware utilization. Fig. 6 depicts the activity factor comparison of unstructured control flow benchmarks between opportunistic early re-convergence and re-convergence at IPDOM point. In our scheme, the warp can re-converge at early re-convergence points in unstructral control flow benchmarks, which leads to higher activity factor.

Fig. 7 shows the activity factor comparison of structural control flow benchmarks between explicit and implicit re-convergence. The activity factor of structural control flow benchmarks gains slight improvement because the redundant re-convergence instructions are removed. Besides, the divergent period of the executed warps are also reduced.

Dynamic instruction count is the number of total instructions executed by the streaming multiprocessors rather than the code size of the kernel. It measures the extra profit that comes from early re-convergence scheme. Fig. 8 shows the different dynamic instruction counts of the explicit PPC method and the implicit PPC method. Since the compiler-generated hint instructions are no longer presented in the implicit scheme, the normalized dynamic instructions in implicit scheme compared to the ones in explicit scheme can be reduced. Instructions executed for the pathfinder benchmark have reduced as much as 7%.

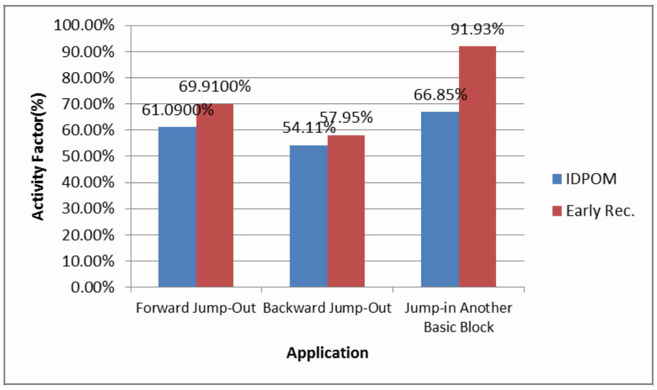


Fig. 6. Activity factor of unstructured control flow benchmarks

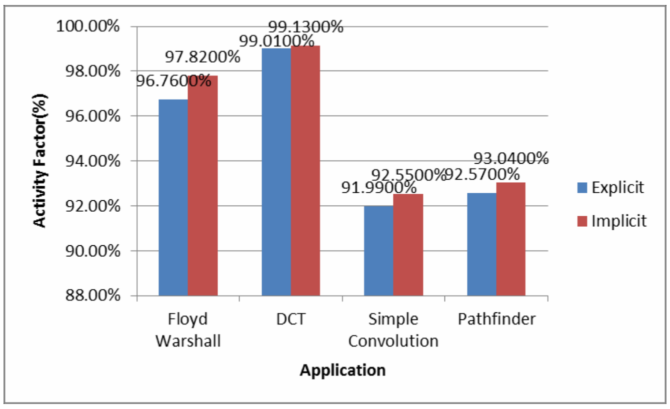


Fig. 7. Activity factor of structured control flow benchmarks using explicit and implicit methods

V. CONCLUSION

In this paper, we propose a novel algorithm that increases the streaming multiprocessor's performance in SIMD divergence execution. Only slight change on existing SM model is needed for implementing the algorithm. The algorithm can be implemented either with or without finalizer's support. In the implicit scheme, the SM gains additional performance improvement owing to avoiding redundant instructions. The proposed implicit scheme eases the design of a finalizer which would typically require to analyze the HSAIL code for the insertion of the explicit instruction at the re-convergence point.

ACKNOWLEDGMENT

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 104-2220-E-006-013.

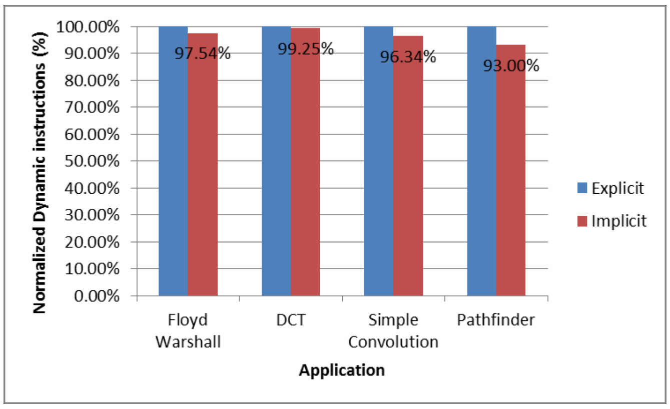


Fig. 8. Normalized dynamic instruction count

REFERENCES

[1] S. Collange, "Stack-less SIMT Reconvergence at Low Cost,"ARENAIRE - Inria Grenoble Rhône-Alpes / LIP Laboratoire de l'Informatique du Parallélisme, 2011.

[2] M. Rhu and M. Erez, "The dual-path execution model for efficient GPU control flow," High Performance Computer Architecture (HPCA), 2013

[3] J. Meng, D. Tarjan and K. Skadron, "Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance," In Proc. 37th Int'l Symp. on Computer Architecture (ISCA), 2010

[4] A. ElTantawy, J.W. Ma, M. O'Connor and T.M. Aamodt, "A scalable multi-path microarchitecture for efficient GPU control flow," High Performance Computer Architecture (HPCA), 2014

[5] AMD SDK: AMD APP Software Development Kit, [Online], Available : http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/.

[6] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," IEEE International Symposium on Workload Characterization (IISWC), 2009.

[7] A. Kerr, G. Diamos and S. Yalamanchili, "A characterization and analysis of PTX kernels," IEEE International Symposium on Workload Characterization (IISWC), 2009.

[8] W.W.L. Fung, I. Sham, G.Yuan, and T.M. Aamodt, "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow," In Proc. 40th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2007

[9] Zhang, F.; D'Hollander, E.H., "Using hammock graphs to structure programs," Software Engineering, IEEE Transactions on , April 2004.

[10] NVIDIA: NVIDIA's next generation CUDA compute architecture: Fermi. Whitepaper, 2010.

[11] AMD: Accelerated Parallel Processing TECHNOLOGY, Evergreen Family Instruction Set Architecture, Instruction and Microcode, Nov, 2011.

[12] HSA foundation: www.hsafoundation.com