영상처리 가속을 위한 CGRA compilation 속도 향상

김원섭, *최윤서, **김재현
삼성전자 DMC R&D Center, *삼성전자 SAIT, ** 삼성전자 DMC R&D Center
wonsub79.kim@samsung.com, *yoons.choi@samsung.com, **jhgim@samsung.com

CGRA Compilation Boost up for Acceleration of Graphics

Wonsub Kim *Yoonseo Choi ** Jaehyun Kim
Samsung Electronics DMC R&D Center *Samsung Electronics SAIT **Samsung Electronics DMC R&D Center

요 약

Coarse-grained reconfigurable architectures (CGRAs) present a potential of high compute throughput with energy efficiency. A CGRA consists of an array of functional units (FU), which communicate with each other through an interconnect network containing transmission nodes and register files. To achieve high performance from the software solutions mapped onto CGRAs, modulo scheduling of loops is generally employed. One of the key challenges in modulo scheduling for CGRAs is to explicitly handle routings of operands from a source to a destination operations through various routing resources. Existing modulo schedulers for CGRAs are slow because finding a valid routing is generally a searching problem over a large space, even with the guidance of well-defined cost metrics. Applications in traditional embedded multimedia domains are regarded relatively tolerant to a slow compile time in exchange of a high quality solution. However, many rapidly growing domains of applications, such as 3D graphics, require a fast compilation. Entrances of CGRAs to these domains have been blocked mainly due to its long compile time. We attack this problem by utilizing patternized routes, for which resources and time slots for a success can be estimated in advance when a source operation is placed. By conservatively reserving predefined resources at predefined time slots, future routings originated from the source operation are guaranteed. Experiments on a real-world 3D graphics benchmark suite show that our scheduler improves the compile time up to 6000 times while achieving average 70% throughputs of the state-of-art CGRA modulo scheduler, edge-centric modulo scheduler (EMS)..

1. INTRODUCTION

CGRAs generally consist of an array of a large number of functional units (FU) connected by an interconnect network. The key challenge in deploying CGRAs is compiler scheduling technology that can effi ciently map software solutions to hardware so that high performance from many FUs can be achieved. To make a good use of CGRA, modulo scheduling [3] of loops is generally employed. There have been research efforts for effi cient modulo scheduling technology for CGRA, which are proven to be useful in application domains, such as audio/video and image processing [1] [2] .

Although the strength of CGRAs is clear in embedded multimedia domains, its long compilation time is a visible weakness, which adversely affects time-to-market cost and developing productivity. Moreover, a slow compilation is one of the main obstacles to wider acceptance of it by other domains. To become a multi-purpose accelerator in various domains including web browsing,gaming, user interfaces and 3D graphics, a fast compilation is a requisite.

To avoid a long compile time, we simplify routing stages in CGRA modulo scheduling so that it can be effectively integrated into the placement step. A placement option is taken only if it is guaranteed not to be invalidated due to later routings. Our scheduler attains this trait by using limited routing patterns called as *qualified* patterns. The patterns have following characteristics such that the availability of all resources within a routing path is quickly verifi able and can cover most of the routing requirements to schedule a DFG.

To the best of our knowledge, our technique is the fi rst CGRA modulo scheduling technique, which can effectively converge to a polynomial time complexity. Our scheduler is faster by 300 times on average, and up to 6000 times within the average performance degradation of 30% comparing to EMS. In some programs, the performance loss is none.

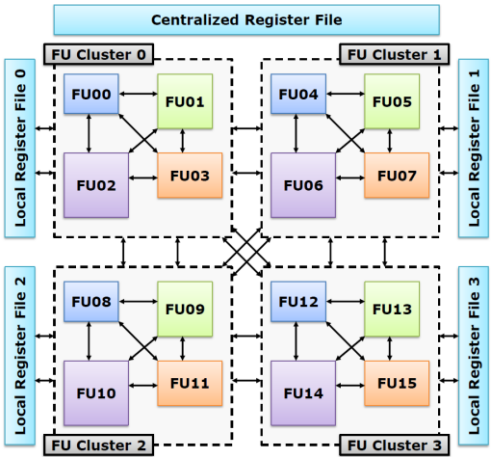

Fig. 1. Target CGRA overview

2. CORE CONCEPTS

Our target CGRA is shown in the **Fig. 1**. The target CGRA has a 4x4 array of heterogeneous FUs, where four FUs are logically grouped into an FU-cluster, organized into four FU-clusters. FU-clusters are connected to each other by a scalable inter-cluster communication network.

An FU-cluster has LRFs that can be directly accessed by FUs in the FU-cluster. An FU can write to a remote LRF through an inter cluster channel (ICC) in its home FU-cluster. An ICC of an FU-cluster is a unidirectional channel through which an FU in the FU-cluster can broadcast its output to all FUs and LRFs in all other remote FU-clusters.

2.1 Qualified Routing Patterns

First, we take advantage of a *direct path* as often as possible. A *direct path* between a source and a destination FUs is defined as the shortest path in terms of the delay of TRNs. It consists only of TRNs without any FU or RF between the two FUs. Second, when a *direct path* is not available, an *indirect path* is utilized. An indirect path contains exactly one LRF between the source and destination FUs. We call the direct and indirect paths defined as above qualified paths. Direct and indirect paths can be subdivided into intra and inter-clusters paths. A path is an *intra*-cluster path if the clusters are the same. Otherwise, the path is an *inter*-cluster path.

Example qualified paths are shown in the **Fig. 2**. The essential resources for each type of qualified paths can be summarized into a fixed set of LRFs and inter cluster channels (ICCs) for each type of qualified path type.

Type	Qualified routing pattern example	Architectural delay (shortest path len.)	Total routing delay	Essential resources	
				LRF	ICC
Intra cluster direct	FU1 → (2) → FU2	d(FU1, FU2) =2	2	-	-
Intra cluster indirect	FU1 → (1) → LWP LRF0 LRP → (2) → FU2	d(FU1, LRF0) =1 d(LRF0, FU2) =2	≥4	*One LRF in dest. cluster*	-
Inter cluster direct	FU1 → (1) → ICC0 → (1) → FU5	d(FU1, ICC0) =1 d(ICC0,FU5) =1	2	-	*One ICC in src. cluster*
Inter cluster indirect	FU1 → (1) → ICC0 → LWP LRF1 LRP → (2) → FU5	d(FU1, ICC0) =1 d(ICC0,LRF1) =0 d(LRF1,FU5) =2	≥4	*One LRF in dest. cluster*	*One ICC in src. cluster*

Fig. 2. Categorization of qualified routing patterns and representative examples

2.2 Conservative Reservation

Our scheduler only uses routes that conform to the qualified paths. One of the benefits qualified patterns offer is that most of the routing requirements for a DFG can be supported by these pattern. Another advantage of qualified paths is that the essential resources and their time slots relative to the source and destination FUs can be determined as shown in the **Fig. 2**. We utilize these characteristics of qualified paths to ensure successful routings originated from an operation in advance when the operation is placed at one FU. More specifically, the resources and their time slots that can guarantee a route are conservatively estimated and reserved before the actual routing step happens.

3. EXPERIMENTAL RESULTS

To evaluate the effectiveness of the proposed algorithm, we took 34 shaders from Taiji benchmark in Basemark ES 2.0. We compared the proposed algorithm, called Fast Modulo Scheduler (**FMS**), to the existing state-of-art CGRA Modulo Scheduler, edge-centric modulo scheduler (**EMS**).

3.1 Compilation Time

Depending on the size of codes, the compile times of **EMS** range from around 1 seconds to 3500 seconds. On the other hand, compile times of **FMS** are less than one second, varying between 0.01 second to 0.9 second. FMS is up to 6074 times faster than EMS, 310 times on average. The compile times of 34 shader codes as their code sizes rise are plotted in **Fig. 3**. Notice that the y-axis is displayed in a logarithmic scale. It is easy to perceive that the compile time of **EMS** is exponentially proportional to the number of operations. As the number of operations increases, the schedule length is generally linearly prolonged, which causes demands for routings with large differences in schedule time. For **EMS**, these long routings are harder to map to CGRA since the search space for them expands exponentially to their lengths. On the other hand, compile times of **FMS** are polynomially proportional to the number

of operations. Overall, FMS satisfy the compile time require-ment for 3D applications, which is not even approachable for existing CGRA modulo schedulers.

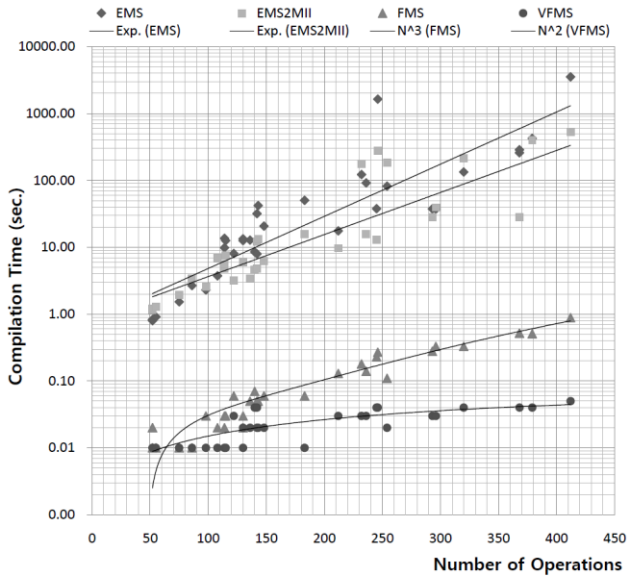

Fig. 3. Comparison of compile time

3.2 Performance

FMS suffers 30% performance degradation on average compared to EMS, which is subdivided into average 22% for pixel and average 37% for vertex shader codes. The performance loss is expected since our scheduler explores much smaller search space by confining routing patterns.

References

[1] B. Mei, S.Vernalde, D.Verkest,H.De Man, and R. Lauwereins, “ Dresc: a retargetable compiler for coarse-grained reconfi gurable architectures,” in Proc. of FPT2002, Dec. 2002, pp. 166- 173.

[2] H.Park, K.Fan, S. A. Mahlke,T. Oh, H. Kim, and H.-s. Kim, “ Edgecentric modulo scheduling for coarse-grained reconfi gurable architectures,” in *Proc. of PACT’ 08*, 2008, pp. 166- 176.

[3] B. R. Rau, “ Iterative modulo scheduling: an algorithm for software pipelining loops,” in Proc. of MICRO27, 1994, pp. 63- 74.