Code Compresion Technique Based on Flexible Bin-Packing Algorithm

Hochan Lee, Bernhard Egger
Department of Computer Science and Engineering
Seoul National University
Seoul, Korea
Email: {hochan,bernhard}@csap.snu.ac.kr

***Abstract*—In this work, we present an improvement of our previous configuration memory compression technique. As the industry requires more flexibility to adapt the fast-changing application grows, coarse-grained reconfigurable architectures (CGRA) are in the spotlight. However, the area and power overhead of the configuration memory to configure the actions for entire hardware entities for each cycle are significant and hinder a broader deployment of CGRA chips. In our previous work, we have presented an efficient and lightweight compression technique of the configuration memory by removing consecutive duplicated lines. To generate compression-friendly code, we apply spatial and temporal optimization in the compiler backend. Here, we suggest an advanced compression technique based on a bin-packing heuristics. The proposed technique is extremely flexible since the technique can predetermine the number of partitions and bitwidth of each partition. In addition, the compressibility of code is improved by applying a more sophisticated temporal optimization. Experiments with 247 kernels from various applications on the commercially used Samsung Reconfigurable Processor (SRP) show a memory reduction of over 75%, on average for individual loops. The compression of unseen code also achieves satisfactory reduction of 45% on average. Compared to previous work, we improve the compressibility by about 10 to 20% for all the benchmark domains.**

I. Introduction

As the industry requires not only application-specific optimization but also the flexibility to adapt the fast-changing applications growth, coarse-grained reconfigurable architectures (CGRA) become an attractive solution that provides both adaptability and energy efficiency while remaining easily programmable [1]. Recently, Samsung Electronics has also adopted CGRA architecture called with the Samsung Reconfigurable Processor (SRP), used in modern TVs, smartphones and imaging appliances such as 4K smart cameras [2] [3].

The CGRA is a design-time reconfigurable processor consisting of many processing elements (PEs) and register files (RFs) connected by an interconnection network comprising multiplexors (MUXs), wires and latches. The PEs are usually heterogeneous and can execute word-level operations on both scalar and vector data. The operations of PEs are executed by accepting the operands from RFs or other PEs through the interconnection network. Unlike traditional ISAs, immediate operands are provided by constant units (CUs). To carry out a correct program, the CGRAs hardware entities are configured in each execution cycle by the pre-defined execution plan stored in the configuration memory. The configuration memory is composed of a number of lines, and each line holds the context to reconfigure all the hardware entities for one cycle of the loop. However, because CGRA has many hardware components, the bitwidth of the configuration memory line can be very wide. Even for a small CGRA with 4x4 PEs, the bitwidth often exceeds 1000 bits. Supporting this long configuration memory line for the current CGRAs which uses frequencies of up to one gigahertz causes a lot of energy consumption and area overhead. This problem hinders a wider deployment of CGRAs and becomes a big challenge for the embedded system market.

To alleviate this problem, we propose a lightweight and high-performance configuration memory compression technique in our previous work [4]. The technique removes consecutive identical lines in the configuration memory. However, since it does not frequently occur [5], our prior technique also applies both statistical temporal and spatial optimization based on the resource sorting algorithm by using edit distance to generate more compression-friendly code.

However, it is not the optimal technique because of following two reasons. First, the prior compression technique must apply the temporal optimization before memory partitioning. Therefore, the temporal optimization generates optimized code for not actually used memory scheme generated from the spatial optimization but the original memory model. Second, it cannot predetermine not only the bitwidth of each partition but also the number of partitions, which hinders the flexibility in architecture design.

In this paper, we propose more efficient and enhanced compression technique based on a bin-packing algorithm. With the basic concept of the previous work, our new technique can dynamically and adaptively apply the temporal optimization while executing spatial optimization process. Therefore, it improves the compressibility for configuration memory. Also, it can freely set the bitwidth of each partition. The proposed compression method achieves a memory reduction of over 70%. In various experiments, newly proposed technique achieved better memory reduction of at least 10% than the previous technique.

In short, this paper makes the following contributions.

- we propose an efficient compression technique that uses advanced memory partitioning based on the bin-packing

algorithm.

- we implement a proposed compression techniques, and apply to the configuration memory of the commercially used CGRA, Samsung reconfigurable processor (SRP).
- we achieve memory reduction of approximately 70% above on the experiments with the 247 loop kernels.

II. Previous Technique

To accomplish the best memory reduction, our previous research proposes and applies two optimization schemes. The first optimization scheme, temporal optimization, tries to make configuration memory lines similar to neighboring lines without influencing correctness. It is achieved by modifying the configuration values of inactive resources which are excluded from the generated schedule. The temporal optimization consists of two steps: **as-soon-as-possible (ASAP)** and **as-last-as-possible (ALAP)**. The ASAP step unconditionally propagates a configuration value generated by schedule to inactive resources as far upward as possible. For example, if configuration value sequence of virtual resource is '1, *, *, 2' and the mark '*' means inactive status, then the ASAP converts to '1, 2, 2, 2'. The second step, ALAP, propagates the preceding configuration value to the inactive resources only if this propagation can achieve better memory reduction than before. It is invoked when all the resources of the same cycle can push down the signal change. If the ALAP keeps propagating even though there exists a resource which cannot propagate a signal change, then it is possible to invoke a one more signal change that decreases the compressibility.

The spatial optimization divides the configuration memory into multiple memory partitions. The important consideration in the spatial optimization is that partitioning is carried out statically. The partitioned memory model once decided at design time cannot be changed. Therefore, it is really important to decide which partition will be encoded by which resources for the compressibility. To alleviate this constraint, in the previous work, we propose a smart memory partitioning technique. The compression technique starts to make a partitioned memory scheme with arranging the resources. The reason for this arrangement is to make having similar configuration patterns with neighboring resources before dividing the configuration memory into several partitions. It decides whether each entity has similar configuration patterns or not by calculating the edit distance among their configuration bitstream. After that, this arranged list of the entities is cut several partitions off yielding the highest memory reductions. Figure 1(a) shows an example of compression process using our previous work. Each row of the table means the configuration memory line. The *i*th line of the uncompressed configuration memory is fetched at cycle *i*. Each value means a configuration value of resources and '*' mark means inactive status. After configuring inactive hardware entities through ASAP-ALAP propagations, the spatial optimization divides the configuration memory into two partitions. As a result, each partition has a potential to remove four lines of the configuration memory in each partition respectively.

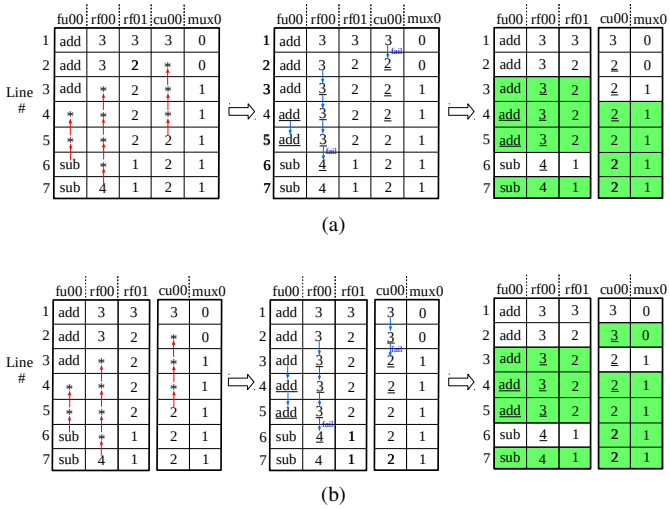


Fig. 1. Improving compressibility by applying temporal optimization dynamically

III. Motivation

Our previous compression technique has room for improvement. The first point is a timing of temporal optimization. The previous technique should apply the temporal optimization before the partitioning because if there exist undecided configuration values then it cannot calculate the edit distance among all the resources. Therefore, the optimization targets to lines of the original configuration memory, not the partitioned memory. The lines of the original configuration memory are completely different from the partitioned memory. It can achieve better compression ratio if the temporal optimization can be applied by using the partitioned memory scheme after the spatial optimization. For example, in the ALAP process of Figure 1(a), cu00 stops the propagation on line 2. This is because the rf01 needs to fetch the new configuration value at line 2. On the contrary, Figure 1(b) applies the temporal optimization after memory partitioning. In this case, it increases one more duplicated configuration memory line in the second partition by configuring to '3'at cycle 2.

The second improvement point is bitwidth of each partition. Since the prior technique just focuses on the memory reduction while divides the configuration memory, it cannot pre-set the bitwidth of each partition. However, this type of memory partitioning has a possibility to invoke a serious problem in the perspective of the architecture design. It is possible to generate memory partition that has extremely small bitwidth even smaller than 50 bit. Applying such memory partition is not possible in reality.

In the next section, we introduce advanced memory partitioning technique based on bin-packing algorithm. This technique is not only eliminating the drawbacks of our previous work, but also achieving enhanced compressibility.

IV. Memory Partitiong based on the Bin-packing

The compression technique by using the bin-packing algorithm builds an efficient memory partition scheme by filling

Fig. 2. Packing Process of compression technique

the predefined number of the partitions with the hardware entities. This technique basically adopts the approach to the compression and decompression of our previous work. On accepting the number of partitions as the input, it starts to compress the configuration memory. The process of memory partitioning based on the bin-packing consists of three steps. The first step is creating an empty partition containers as much as the predefined number of partitions. All the partitions are empty and will be filled with the hardware entities.

Next, the technique tries to find proper partitions for each resource. This step is called to *Packing*. Packing is conducted through sequentially traversing all hardware entities. An order of the visit is decided by shuffling. When the technique visits an entity, it decides a proper memory partition based on the potential of memory reduction. It calculates the memory reduction of each partition if the entity is inserted. However, in this case, it is faced with the same problem of our previous work. In this case, it is impossible to calculate the memory reduction of each partition because of inactive entities. The inactive entities do not have configuration value and it hinders the decision whether a line of the configuration memory can be eliminated or not. To solve this problem, the compression technique dynamically applies the temporal optimization. At the final step, the compression technique chooses the partition which gives the highest memory reduction with the current entity. The configuration of the entity will be encoded to the chosen memory partition.

Figure 2 shows an example of the bin-packing process. The depth of the configuration memory for target loop kernel is '5' and the number of memory partitions is two. After shuffles the hardware entities, the technique traverses these entities iteratively to find an appropriate partition. In Figure 2 (a), the technique tries to find an appropriate partition for PE03. At first, it temporarily includes PE03 to all the partitions like Figure 2 (b). Then, the temporal optimization starts both ASAP and ALAP propagation processes to configure all the inactive resources. Figure 2 (c) describes a result of the temporal optimization. The green boxes represent consecutive duplicated memory lines that can be removed from the configuration memory. The partition 0 and partition 1 have three and two of compressible lines respectively. If we assume that all the resources use one bit for the configuration, then the partition 0 can reduce *(1*4)*3=12bits* and the partition 1 can reduce *(1*3)*2=6bits*. The partition 0 can achieve better memory reduction than the partition 1. Therefore, the configuration memory of the PE03 is encoded into the partition 1. Figure 2 (d) shows the result of the packing process for PE03. The technique visits the next entity, RF02.

The important advantage of the proposed technique is that it can optionally predetermine the bitwidth of each partition. If there is predetermined bitwidth information for partitions, then the algorithm limits the number of hardware entities of each partition. Only a partition that is smaller than or equals to the predefined bitwidth can be encoded the configuration of hardware entities.

V. Experiments

A. Environment and Benchmarks

In this section, we describe our experimental results of the configuration memory compression technique that uses memory partitioning based on the bin-packing algorithm. We have used the Samsung Reconfigurable Processor (SRP) [6] that is the commercial CGRA to implement the proposed technique.

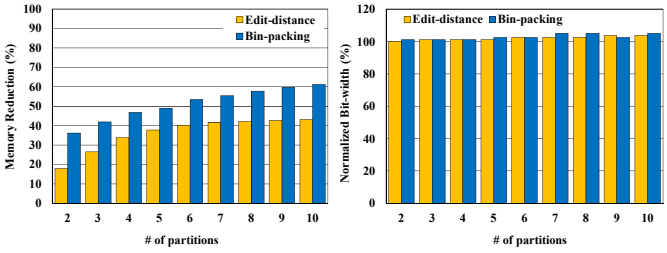
The benchmarks consist of thirty-two real world applications. A total number of the loops contained all the applications are 247 and the number of lines of the configuration memory generated by the compiler is 1978. The detailed information of benchmarks is described in Table I. Groups with the (SIMD) suffix use special instructions for optimizing the respective application domain.

B. Memory Reduction Comparison

On the whole case, the newly proposed compression technique using bin-packing algorithm shows better memory reduction than the previous technique using edit distance. Figure 3 shows memory reduction according to the number of partitions of both techniques by using the entire 247 kernel loops as inputs. The bin-packing technique improves memory

TABLE I
BENCHMARKS

Application Domain	Applications (# kernels)	Total kernels	Total conf. lines
Graphic	3D (9) matrix (3) opengles_r269 (9) PICKLE_V1.2 (14)	35	155
Video	aac.1 (16) avc.swo (11) EaacPlus.1 (23) mp3.1 (10) mpeg_surround (26)	86	588
Voice	FIR (1) huffman_decode (6) bit_conversion (1) histogram (1) amr-wbPlus (12)	21	174
Voice (SIMD)	FIR (1) high_pass_filter (1) huffman_decode (5) bit_conversion (1) histogram (1)	9	116
Resolution (SIMD)	bilateral (3) gaussian_smoothing (3) optical_transfer_function (6)	12	101
Imaging (SIMD)	csc (1) dct (1) median (1) sad (1) gaussian_filter (19)	23	391
Face detection	face detection (35)	35	243
Others	word_count (13) merge_sort (5) bubble_sort (3) array_add (5)	26	210
Total	**32**	**247**	**1978**



(a) (b)

Fig. 3. Results per the number of partitions

reduction average of 15% compared to edit distance technique with almost same bitwidth.

Next, we compare the memory reduction per target benchmark domain between two techniques with four partitions. Table II describes the detailed memory reduction information. The bin-packing technique accomplishes good memory reduction more 10% on average than the previous technique.

TABLE II
MEMORY REDUCTION PER BENCHMARK DOMAIN

Target Domain	Edit distance (%)	Bin Packing (%)
Image processing (SIMD)	19.45	29.40
Resolution (SIMD)	27.77	38.11
Face Detection (SIMD)	40.16	45.1
Voice processing	48.15	57.04
Grpahic	53.44	60.43
Video	52.81	65
Voice processing (SIMD)	43.05	65.71
Individual apps	62.25	70.67
Individual loops	69.61	75.83
Entire loops	33.88	55.07

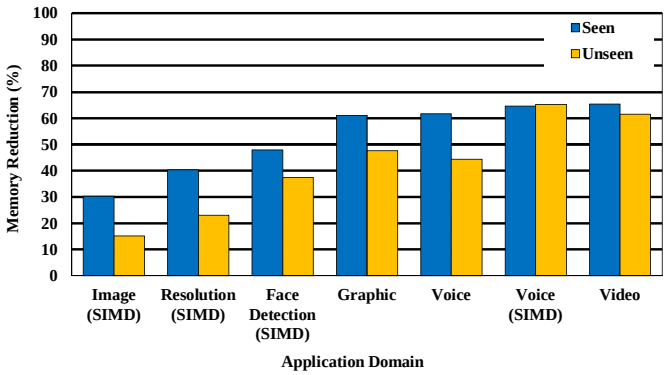


Fig. 4. Memory reduction for trained and new loop kernels

C. Compressibility of trained vs. new loops

Since a big advantage of CGRA is reconfigurability, it is very important to achieve great compressibility even for the new target loops. To verify the scalable compressibility, we train loop kernels 80% of each application domain and execute configuration memory compression with the rest 20% loop kernels.

The results are presented in Figure 4. We successfully achieve a great scalability. The memory reduction of the new loops is only decreased 15% on average compared to the trained part.

VI. CONCLUSION

We presented an advanced code compression technique based on memory partitioning that employs a bin-packing heuristics. The technique removes the drawback of prior work and accomplishes very good compressibility of code. We verify the proposed technique through a wide range of experiments on the SRP, a commercial CGRA processor. For 247 loop kernels from various applications, the average memory reduction is over 75% above. Compared to the previous technique, compressibility is improved by 10-20% on average.

REFERENCES

[1] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix," in *International Conference on Field Programmable Logic and Applications*. Springer, 2003, pp. 61–70.

[2] J. Lee, Y. Shin, W.-J. Lee, S. Ryu, and J. Kim, "Real-time ray tracing on coarse-grained reconfigurable processor," in *Field-Programmable Technology (FPT), 2013 International Conference on*. IEEE, 2013, pp. 192–197.

[3] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim, "ULP-SRP: Ultra low power Samsung Reconfigurable Processor for biomedical applications," in *Field-Programmable Technology (FPT), 2012 International Conference on*. IEEE, 2012, pp. 329–334.

[4] B. Egger, H. Lee, D. Kang, M. S. Moghaddam, Y. Cho, L. Y. S. Kim, S. Ha, and K. Choi, "A space- and energy-efficient code compression/decompression technique for coarse-grained reconfigurable architectures," in *Code generation and optimization (CGO), 2017 International Conference on*. IEEE, 2017.

[5] N. Aslam, M. J. Milward, A. T. Erdogan, and T. Arslan, "Code compression and decompression for coarse-grain reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 12, pp. 1596–1608, 2008.

[6] D. Suh, K. Kwon, S. Kim, S. Ryu, and J. Kim, "Design space exploration and implementation of a high performance and low area coarse grained reconfigurable processor," in *Field-Programmable Technology (FPT), 2012 International Conference on*. IEEE, 2012, pp. 67–70.