FAReP: Fragmentation-Aware Replacement Policy for Task Reuse on Reconfigurable FPGAs

Godwin Enemali, Adewale Adetomi, Tughrul Arslan
Institute for Integrated Micro and Nano Systems
School of Engineering, The University of Edinburgh
Edinburgh, UK
{g.enemali, a.adetomi, t.arslan}@ed.ac.uk

***Abstract*—The use of reconfigurable chips such as FPGAs in embedded systems for many runtime applications is limited by large reconfiguration time. Techniques to circumvent this limitation relies on hardware task reuse which preserve certain circuits on the chip. However, the frequent addition and removal of circuits while preserving others on the chip will inevitably lead to fragmentation of its area, in an ongoing manner. In this paper, we present a fragmentation-aware replacement policy (FAReP) for reusing tasks on reconfigurable chips. FAReP aims not only at circumventing reconfiguration time, but also offering some defragmentation of the chip area at no extra reconfiguration cost. Our results show that FAReP leads to a reduced task rejection ratio, at least a 13% reduction in average unused chip area and up to 29% of reconfiguration time could be avoided compared to state of the art techniques based on task reuse.**

Keywords—Reconfigurable hardware; hardware Task reuse; caching; replacement policy; configuration reuse

I. INTRODUCTION

Embedded systems are increasingly required to perform complex operations with strict area and energy constraints. This is often the case in embedded systems. In order to meet these constraints, reconfigurable hardware such as FPGAs are used as computation platform support for these systems [1]. FPGAs now include dynamic partial reconfiguration capability, and therefore needed circuits could be (re)configured to perform the desired tasks, after which they could be replaced by other circuits at runtime. This leads to better area utilizations as different tasks could be swapped in and out when needed [2].

However, the reconfiguration duration of state of the art FPGAs is quite significant, often in the order of milliseconds [1]. This is not well amenable to the runtime scenario of most embedded systems requiring frequent context switches of its tasks [3]. To cope with this, *hardware task reuse* have been proposed [1] [4] [5] [6]. Its aim is to preserve tasks with high configuration duration on the chip even after their execution, if they are likely to be required again *soon*. However, the frequent addition and removal of circuits while preserving others on the chip will, very often, lead to fragmentation of its area, in an ongoing manner. Ongoing fragmentation happens even when individual tasks are carefully well-placed on their initial arrival [7].

Most of the existing task reuse schemes do not address the problem of ongoing fragmentation, and thus leads to low utilization of the chip. At the heart of any task reuse technique (akin to software caching mechanisms) is its *replacement policy* – the choice of which task(s) to preserve on the chip. An inefficient replacement policy would lead to an overall increase in configuration time and task rejection [8].

Ongoing fragmentation would have been grossly reduced by defragmentation, which involves a time-to-time rearrangement of tasks on the chip [8]. However, large reconfiguration time makes such defragmentation almost infeasible on the current FPGA architecture.

In this paper, we present a reuse scheme based on a novel replacement policy, FAReP. Unlike any other task circuit reuse schemes, our scheme includes a form of *defragmentation* at no extra reconfiguration cost. The second key contribution of this paper is the presentation of a novel fragmentation metric for heterogeneous chips. We note that most existing techniques for quantifying fragmentation on reconfigurable chips are well suited to homogenous ones. They use the assumption that the chips only consist of same resource type – mostly reconfigurable logic blocks (CLBs). Hence they are not directly applicable to heterogeneous chips.

II. RELATED WORKS

Most earlier works in task reuse are targeted only at circumventing reconfiguration cost, without addressing ongoing fragmentation. In addition, they only address applications suitable for homogenous chips. Three notable ones are highlighted below.

The work in [1] presented a task reuse strategy in which tasks with low *criticality* values and occur furthest (in the available set of schedule) have less chance of being preserved on the chip. The strategy does not consider any ongoing fragmentation issues, and demands good prior knowledge of the execution order of tasks on the hardware. It is based on a slotted architecture which often leads to internal fragmentation, and is only suited for tasks which can fit in similar slots.

In [6], a number of replacement policies were surveyed. This included *least recently used*, *penalty based algorithm* (based on reconfiguration time and frequency of use) as well as others suited only for static reconfiguration chips. Their performance on different FPGA models was also evaluated, and it was reported that the *penalty based algorithm* had better performance. The work did not address any ongoing

978-1-5386-3408-0/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPSW.2017.153

fragmentation, but rather proposed a new FPGA architecture that would suit defragmentation.

The work in [9] used a penalty based algorithm replacement technique which was called Reconfiguration-to-Execution Ratio (RER). To choose a candidate to be replaced, they compute an RER value for each potential candidate as the ratio of its reconfiguration time to execution time, multiplied by execution frequency. The candidate with lowest RER value is replaced first. The technique does not address any ongoing fragmentation issues and is only suited for homogenous chips.

A reuse based scheduling (RBS) algorithm is presented in [5]. The replacement scheme used was called Least Probability of Recurrence (LPR). To deal with fragmentation, the platform was divided into two resizable slots – one reserved for significant tasks (which are preserved) and the other for non-significant tasks (which are not preserved). Apart from being computationally intensive, the use of slots often leads to internal fragmentation.

III. THE FRAGMENTATION AWARE REPLACEMENT POLICY ALLOCATION SCHEME

It is worth noting that in our scheme, the mechanism of communication among tasks is achieved via the Internal Configuration Access Point (ICAP) as described in [2]. Hence overlay of tasks to specific partition pins is not necessary to establish communication. Tasks can be placed in any matching position by changing appropriate frame addresses in their bitstreams. In addition to minimizing chip fragmentation, FAReP aims to free up the ICAP by reducing configuration time, and thus make it more available for the task of communication. In the following sub-sections, the placement flow of which FAReP forms an integral part is described.

A. Offline Stage

Two preparations are applied to the tasks in their offline stage. The first is to improve their place-ability on the chip at runtime [10]. For tasks whose dependency with other tasks is unknown at design time, we chose a synthesis position to maximize the total number of such positions present on the chip. For tasks whose dependencies are known at design time and which are likely to share chip resources at overlapping times, we select synthesis positions such that a maximum number of non-overlapping positions, which are as evenly distributed as possible is achieved.

The second step is to prepare tasks' interface for communication during runtime. Communication is basically done by copying the contents of a producer task's output stored in its output results buffer (ORB) to the input data Buffer (IDB) of a consumer task. We used the routines provided in [11] for wrapping the tasks.

This work has been supported by the Nigerian government under the Presidential Scholarship Scheme for Innovation and Development (PRESSID)

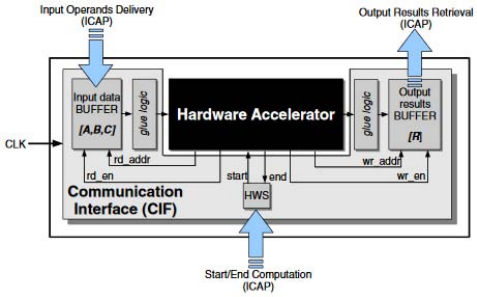

Fig. 1. Overview of Task Wrapping for ICAP based communication [11]

Figure 1 shows a wrapped task as per this format. In the figure, Hardware Accelerator refer to the standard task without the wrapping. The IDBs and ORBs are implemented using BRAM(s) (for a large data producing/consuming task) or LUT-RAM(s) for tasks dealing with small data. The Hardware Semaphores (HWS) are provided for signaling the start/finish of the task execution. The glue logic defines the connection between the standard task and the input/output data buffers. It occupies only 25FF and 35 LUTs for a 32-bit interface in our implementation. This is in addition to the memory requirements of the task. Data is obtained from (written to) a task by reading (writing) the contents of the ORB (IDB) using the (ICAP).

B. Runtime Stage

On arrival of a new task (NT), the placement scheme tries to place the task. There are 3 possible outcomes: first, it could be assigned to an idle circuit if any is found capable of executing the task. This is possible as circuits are not removed from the chip until the resources they hold are required by another task. The second option is to queue it, waiting on a computing task. This is possible if a suitable instance is configured on the chip but is actively involved in computation, and would become free in time for the execution of the NT without violating its deadline requirements. This possibility is checked by verifying that the wait time, t_w of the new task (equation 1) is less than the remaining computation time, t_r of the busy instance (equation 2). Ø is the computation time of the placement routine, t_{curr} is the current time and t_s the time when the computing instance started its current computation. These two options leverages task reuse, and configuration time is circumvented, freeing the configuration engine for other activities.

$$t_w = (d - t_{curr}) - (e + \emptyset) \quad (1)$$

$$t_r = e - (t_{curr} - t_s) \quad (2)$$

The third possible option in the placement of NT is to scan for a new location for the task on the chip. To determine an initial placement, a scan function obtains the set of available locations. For each of these positions, a *fragmentation coefficient* (FC) is computed. The details of computing FC is given below. The available location with the least FC is chosen

for placement of the NT. If all three of these options fail to allocate a position to the task, the replacement scheme explained in section III.B.2 is triggered.

1) Quantifying Fragmentation Coefficient

Many state-of-the-art techniques for computing fragmentation such as the adjacency or vertex based heuristics [7] do not suit a heterogeneous chip (which is the target in our case). This is because tasks' location on heterogeneous chips have definite start and end points which often do not fall at the border of existing placements. Our approach, is based on computing the 'stand-alone-ness' (as well as the adjacency – if it exists) of the area (to be) occupied by the task(s). This is done using equation 3, which computes the average stand-alone-ness in the horizontal and vertical direction.

In equation 3, d_{fh} and d_{bh} (d_{fv} and d_{bv}) refer to the average distance – in number of CLB cells – in the forward and backward horizontal (vertical) directions. r_h (r_v) is the range of the distances, and α is chosen to give appropriate relative significance between the distance and range terms. It was determined to be 3 by exhaustive simulations using varying task sizes and placement positions on the xc7z100 FPGA chip. k is a normalization factor chosen to reflect the relative difference between the horizontal and vertical resolution of the cells on the chip. It was chosen to be 50 for the Xilinx's 7 series chip as 1 row consists of 50 CLBs. The parameters m and n are the horizontal and vertical dimensions of the chip.

$$FC = \frac{1}{2}\left[\frac{\left((d_{fh} + d_{bh})/m\right)^{\alpha}}{r_h} + \frac{(k.(d_{fv} + d_{bv})/n)^{\alpha}}{r_v}\right] \qquad (3)$$

As an illustration, consider figure 2 which shows different areas for tasks on a chip. For the chip shown, m = 10, n = 8 and k = 1. We used α = 3. Table 1 shows the fragmentation coefficients for the scenarios shown in figure 2 using our technique. As shown, Task 1 has the least FC (0.057) maintaining the least stand-alone-ness and having the highest contact at its borders. Task 4 has the highest FC as shown. Note that although both tasks 1 and 4 have the same adjacency at their borders, their FC values are very different, hence adjacency may not be a good metric for fragmentation on heterogeneous chips.

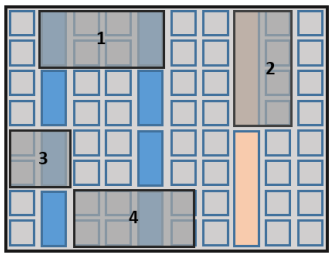

Fig 2: Task Areas on a Chip

TABLE I: FRAGMENTATION COMPUTATION

Tasks	Fragmentation Coefficient		
	Horizontal	*Vertical*	*Resultant*
1	0.025	0.032	0.057
2	0.025	0.046	0.071
3	0.000	0.139	0.139
4	0.180	0.046	0.226

2) The Basic Idea of FAReP

The idea of FAReP is this: for a set of idle circuits with *comparable* reconfiguration cost, those that contribute more to the fragmentation of the chip are less likely to continue to be preserved on the chip. We define reconfiguration cost as product of configuration time and frequency of usage of a circuit instance. We quantify the term comparable below.

FAReP relies on three parameters to determine which of the idle instances to replace: configuration time, c, likelihood of re-use, and their degree of fragmentation. The configuration time of a circuit is computed based on our custom configuration controller and stored in addition to the task's other parameters. We measure the likelihood of re-use of an instance using its execution history. We maintain a parameter, number of reuse (NU) for each configured instance. This parameter is initialized after (re)configuration, and incremented each time the instance is used to execute a task. The cost of reconfiguration, ψ, is computed as: $\psi = c\ x$ NU for all idle instances which could be potentially replaced to place an incoming task. Now, we define a threshold α and state that all instances whose differences in ψ is less than α have *comparable* reconfiguration cost. The instance of these with the highest FC is chosen to be replaced ahead of others. The effect of α values is discussed in the result section.

Figure 3 shows an example that points outs the benefit of our replacement policy. A set of tasks (A – E) are requested to be executed on the chip twice, in the order A to E with the configuration time of B and C assumed to be comparable, but with B's slightly lower. The figure shows a comparison between FAReP and that based on reconfiguration cost and Execution Rate (RER). Each stage represents a new task placement (and removal of another if necessary). During the first execution cycle, the three initial placements (A to C) are the same for both schemes as there is no need for any replacement. For stage I_4, to place D, FAReP (figure a) chooses to replace C because its FC (computed with equation 3) is 0.1 as against 0.08 for B. However, RER (figure b) chooses B since its reconfiguration time is smaller than that of C. The advantage of replacing instance C is that both D and E can be configured onto the chip without having to remove B in addition as is the case with the RER. This is because the removal of C made other fragmented spaces around it useable. In the second cycle of execution (after t_1), only two configurations are required for FAReP, those of Tasks C and D. RER requires 4 configurations (B, C, D and E).

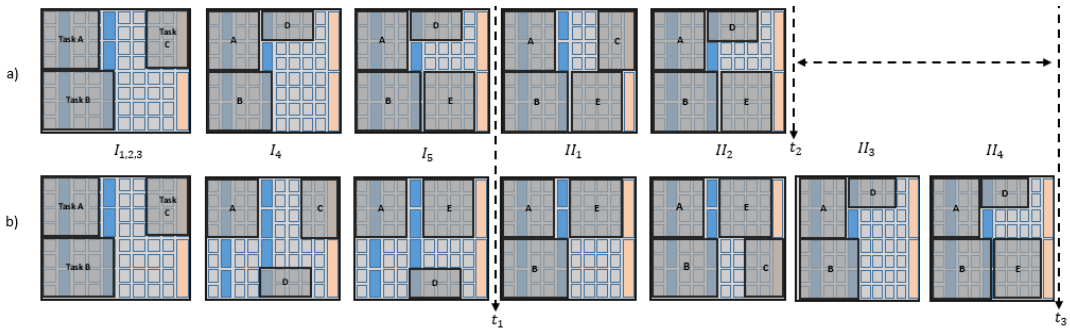

Fig. 3: Comparison of replacement Schemes for the execution of five tasks on a chip a) FARePolicy b) RER policy

IV. EXPERIMENTAL SET UP

To test the performance of our replacement policy, we created a standalone simulation framework on an intel(R) Core™ i7 processor, running at 3.40GHz. We also implemented the RER scheme on the same platform, and estimated results for RBS scheme. The reconfigurable platform simulated was that of the Programmable logic of the Xilinx's xc7z100ffg900-2 chip. Each row is made up of 134 columns, consisting of 12 Block RAM (BRAM) columns, 15 Digital Signal Processor (DSP) columns and 107 Configurable Logic Block (CLB) columns. It has 7 rows. Four sizes of FPGAs were considered corresponding to 25%, 50%, 75% and 100% of the chip which labelled 1 to 4 respectively in figure 4.

The tasks used for the simulation were based on utilization and estimates of execution time of common hardware tasks, obtained from [12]. 20 sets each consisting of 100 tasks were generated. Tasks are placed as soon as they arrive and the allocator is ready. If more than one task arrive at the same time, an Earliest Deadline First (EDF) scheme in [2] is used.

V. RESULTS AND DISCUSSION

The result of our scheme was compared with 2 other major hardware task re-use schemes. These are the RER in [9] and RBS in [5]. Our comparison is based on: task rejection ratio (TRR), average unused area at task rejection (AUATR), and Average Configuration Clock Cycles Saved (ACCCS). We define TRR as the ratio of the number of task rejections to the total of placement requests; AUATR= ratio of sum of unused area when task rejection due to area occurs to number of tasks rejected for lack of area. Finally, ACCCS refers to the sum of the configuration clock cycles of all tasks which reused idle instances, and hence did not have to be configured.

Figure 4 shows the variation of the TRR for RER, FARePand RBS. FARePis evaluated for various values of α as discussed in III.B.2. For the results shown, α is computed as 10%, 20%, 40%, 80% and 100% of the difference between the least and the largest configuration times of the tasks. As shown, the performance of the schemes is quite similar for very small and large chip areas (i.e. 1 and 4). This is because the chance of preserving tasks on small chips is very low as the limited area is often occupied by computing tasks. Also, a large chip area can preserve many tasks and need for replacement is rare.

For the medium-sized chips, FARePwith α = 0.1 has the least TRR of 6%, compared to the 10% and 11% respectively for RER and RBS. The successive degrading performance of FARePwith increase in α is due to the loss in the significance of configuration cost with increasing α. This conforms with [6] and [9] which state that configuration time and frequency of use of any instance are major factors in any replacement policy.

Table II some other performance data. As shown, FARePsaves the configuration engine over 1200 clock cycles compared to RER. However, its computation time at 100MHz is slightly larger than that of RER. This is due to extra computational steps required to compute fragmentations. However, it is faster than RBS. The average wasted area when a task is rejected is lower for FARePthan RBS and RER. This is precisely due to the defragmentation offered by FARePe

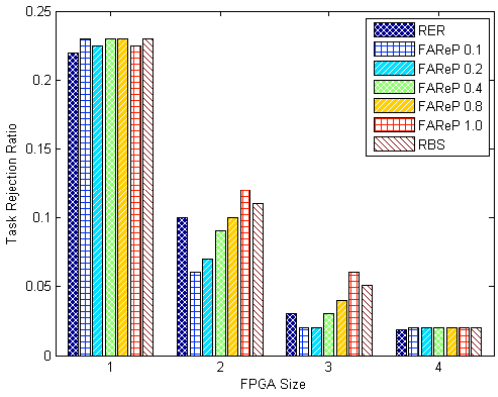

Fig. 4: Variation of the Task Rejection Ratio with various replacement schemes on different sizes of FPGAs

TABLE II. COMPARISON OF SOME PERFORMANCE PARAMETERS OF VARIOUS PLACEMENT SCHEMES

	RER	**RBS**	**FAReP**
ACCCA[a]	4069	3426	5277
Normalized Runtime of Scheme (µs)	0.86	1	0.94
AUATR[b]	0.95	1	0.82

[a]Average Configuration Clock Cycles Saved

[b]Normalised Average unused area at task rejection

VI. CONCLUSION AND FUTURE WORK

In this work, we have presented a replacement scheme for efficient tasks reuse on reconfigurable chips. FAReP is aimed not only at circumventing reconfiguration time, but also offering some defragmentation of the chip area at no extra reconfiguration cost. Our results show that FAReP leads to a reduced task rejection ratio, at least a 13% reduction in average unused chip area and up to 29% of reconfiguration time is avoided compared to state of the art techniques based on task reuse. As a future work, we hope to integrate FAReP into a full placement management circuit.

REFERENCES

[1] J. A. Clemente, J. Resano, C. Gonzalez, and D. Mozos, 'A Hardware Implementation of a Run-Time Scheduler for Reconfigurable Systems', *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 19, no. 7, pp. 1263–1276, Jul. 2011.

[2] X. Iturbe *et al.*, 'R3TOS: A Novel Reliable Reconfigurable Real-Time Operating System for Highly Adaptive, Efficient, and Dependable Computing on FPGAs', *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1542–1556, Aug. 2013.

[3] A. DeHon and S. Hauck, *Reconfigurable Computing: Theory and Practice of FPGA based computation*. Amsterdam: Morgan Kaufmann, 2008.

[4] A. Morales-Villanueva, R. Kumar, and A. Gordon-Ross, 'Configuration prefetching and reuse for preemptive hardware multitasking on partially reconfigurable FPGAs', in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 1505–1508.

[5] M. Mansub Bassiri and H. Shahriar Shahhoseini, 'Configuration Reusing in On-Line Task Scheduling for Reconfigurable Computing Systems', *J. Comput. Sci. Technol.*, vol. 26, no. 3, pp. 463–473, May 2011.

[6] Z. Li, K. Compton, and S. Hauck, 'Configuration caching management techniques for reconfigurable computing', in *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, 2000, pp. 22–36.

[7] J. Tabero, J. Septién, H. Mecha, and D. Mozos, 'Allocation heuristics and defragmentation measures for reconfigurable systems management', *Integr. VLSI J.*, vol. 41, no. 2, pp. 281–296, 2008.

[8] K. Compton, Z. Li, J. Cooley, S. Knol, and S. Hauck, 'Configuration relocation and defragmentation for run-time reconfigurable computing', *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 10, no. 3, pp. 209–220, Jun. 2002.

[9] K. Sigdel, C. Galuzzi, K. Bertels, M. Thompso, and A. D. Pimentel, 'Runtime task mapping based on hardware configuration reuse', in *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, 2010, pp. 25–30.

[10] M. Koester, W. Luk, J. Hagemeyer, and M. Porrmann, 'Design optimizations to improve placeability of partial reconfiguration modules', in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2009, pp. 976–981.

[11] X. Iturbe, K. Benkrid, T. Arslan, R. Torrego, and I. Martinez, 'Methods and Mechanisms for Hardware Multitasking: Executing and Synchronizing Fully Relocatable Hardware Tasks in Xilinx FPGAs', in *2011 21st International Conference on Field Programmable Logic and Applications*, 2011, pp. 295–300.

[12] 'Projects :: OpenCores'. [Online]. Available: https://opencores.org/projects. [Accessed: 16-Jan-2017].