Area Reclamation Strategies and Metrics for SRAM-Based Reconfigurable Devices

Abdel Ejnioui and Ronald F. DeMara
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, Florida 32816, U.S.A

Abstract— Defragmentation is a fundamental resource management service allowing Reconfigurable Computing Systems (RCSs) to efficiently utilize resources when tasks are dispatched dynamically. Only well orchestrated interactions between the components of the reconfigurable resource management system can sustain the highest possible performance level for applications running on these RCSs. While scheduling and placement have been extensively studied, defragmentation and its impact on overall system performance is still not well understood. This paper quantifies factors related to defragmentation that can affect performance in terms of level sustainement. The paper concludes by proposing an experimental approach to study this problem.

Kewords: Defragmentation, Reclamation, FPGA, Sustainment.

1.0 Introduction

A decade ago, *reconfigurable computing* emerged as a new paradigm suitable to address the processing needs of many compute-intensive applications. Today, most RCSs consist of clusters of commercial off-the-shelf components such as microprocessors and memories augmented with FPGA chips. Figure 1 shows a sample organization of an RCS. The key to exploiting an RCS is the virtualization of hardware whereby the fabric of a reconfigurable device can be reused ad-infinitum to execute many computations concurrently subject only to area and performance constraints. Recently, several efforts went into developing software environments that ease the compiling process of the application on RCSs [1]. Other efforts went further by proposing operating systems fitted to manage the hardware resources of an RCS [2]. While there are arguments for and against the development of operating systems for RCSs, their primary benefit stems from their support for multi-tasking. Multi-tasking requires facilities to schedule and place the computational tasks onto the reconfigurable fabric of the FPGA chips embedded within the RCS. In general, the operating system of an RCS provides services to support the dispatching of computation tasks to the FPGA chip in order to accelerate the running applications. These services consist of scheduling the tasks, placing the tasks on the FPGA, and performing defragmentation if task placement fails. In addition, the operating system can provide services to control configuration swapping in and out of the FPGA chip for the purpose of supporting task placement and defragmentation [3]. Typically, while an application is executing on an RCS, computational tasks are dispatched from the application via the general purpose processor to the FPGA board at various times. These tasks can be generated at runtime or pre-stored in an auxiliary memory for later usage. In an RCS, tasks are downloaded to the FPGA chip by placing them on the areas of the chip that are not occupied by other tasks. Tasks are assumed to have rectangular shapes with fixed orientations. After a task completes its execution, it is purged from the reconfigurable fabric of the chip. As the application continues its execution, tasks are added and deleted in a dynamic fashion leaving ultimately the reconfigurable fabric of the chip highly fragmented. Although, at times, the total unoccupied area in the fabric may be larger than the area needed to place the incoming task, severe fragmentation can eliminate contiguous areas in the fabric that are sufficiently large to accommodate the placement of the incoming task. Several placement studies have been previously undertaken in which different algorithmic approaches for task placement are proposed [1, 4-6]. Regardless of how efficient task placement can be, the reconfigurable fabric will eventually reach an advanced fragmented state where task placement becomes extremely difficult. Unless the tasks already placed on the chip are moved and compacted as fast as possible, task placement cannot be

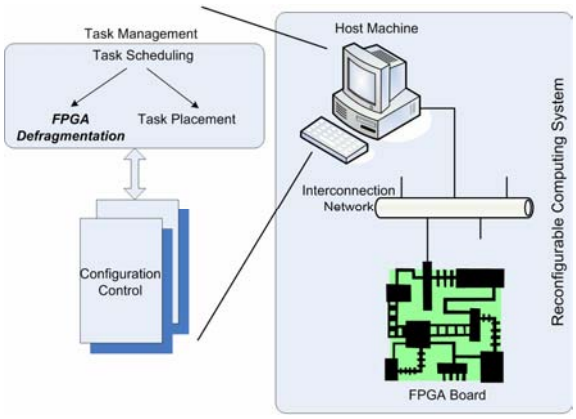

Figure 1. Example of a reconfigurable computing system.

performed, and subsequently application performance cannot be sustained at the same level. To reach this goal, defragmentation has to be performed in the most efficient manner without severely disrupting the progress of the application execution. In fact, defragmentation should aim at bringing the overall performance back up to its previous level before the fragmentation of the FPGA chip has reached a severe level. Whereas task scheduling [1, 7] and placement [1, 6] [4, 5] have been studied in depth, no significant studies have been published to understand the impact of defragmentation on application performance [8]. Such studies could ultimately help in gaining meaningful insights on the interplay between defragmentation, placement and scheduling.

The paper is organized as follows. Section 2 presents the primary defragmentation issues that need to be addressed in order to achieve performance sustainment while section 3 describes the research needed to understand the primary issues surrounding defragmentation. Section 4 concludes the paper.

2.0 Performance Issues in Defragmentation

It is imperative that defragmentation be examined from a global system perspective in order to understand how it can sustain performance. To help develop a new theory for defragmentation, the following issues need to be addressed:

(i) How can the fragmentation of an FPGA chip be accurately quantified and measured?
(ii) What is the best time to perform defragmentation?
(iii) Where in the reconfigurable fabric of the FPGA chip can defragmentation be performed?
(iv) How much of the reconfigurable fabric of the chip need to be defragmented at a time?
(v) How can defragmentation be performed on an FPGA chip?
(vi) If defragmentation is feasible, what is its place in the system architecture of the RCS?

2.1 Quantification of Defragmentation

A well-targeted fragmentation strategy needs a reliable measure to assess the severity of the fragmentation of the reconfigurable fabric of an FPGA chip. For instance, storage systems, such as primary memory and disk drives, have been using, for some time, various fragmentation metrics to assess the state of fragmentation of their hardware resources. These metrics are used to determine when and how much defragmentation is needed. One can extend this approach to using the fragmentation metric as the basis to guide the defragmentation process on the reconfigurable resources of the RCS. In fact, an accurate fragmentation metric can help answer questions (ii), (iii), and (iv) listed above. Based on [8], one can view the reconfigurable fabric of an FPGA chip as a square area containing an array of smaller empty square areas called *cells*. In the context of FPGA chips, cells are equivalent to reconfigurable logic blocks (CLBs). After repetitive addition and deletion of tasks to and from the FPGA, the reconfigurable fabric of the chip becomes fragmented as shown in Figure 2. Figure 2 shows tasks T_1 and T_2 occupying two and six cells respectively. The incoming task T_3, consisting of six cells, cannot be placed on the chip although there is sufficient room left on the FPGA. At first examination of this example, one is tempted to use the empty area as a measure of fragmentation. However, this approach is too simplistic to account for the two dimensional properties of fragmentation. To illustrate, Figure 3 shows two FPGA chips with equal empty area and different fragmentation states. It is clear that placing the same incoming task T_3 shown in Figure 2 on a chip with fragmentation state 1 shown in Figure 3 is totally

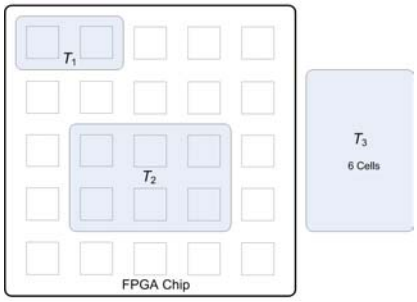

Figure 2. Fragmentation of an FPGA chip.

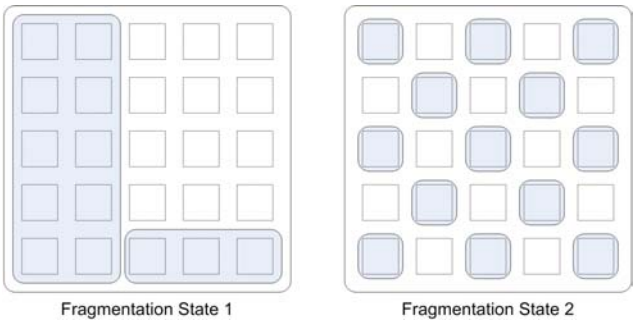

Figure 3. Two different fragmentation states with equal empty areas.

different than placing it on a chip with fragmentation state 2. In fact, a good metric can help narrow the semantics of fragmentation in the sense that a highly fragmented chip ought not to give any indication on how full or empty it is. It should instead give an indication on the degree of scattering of the holes (or empty areas) across the chip area. In this context, a highly fragmented chip is a chip which contains a number as large as possible of holes separated by as many occupied areas as possible. In this case, the degree of dispersion of the holes is at its maximum (100%), and subsequently the chip is neither completely empty nor completely full. To get a large number of scattered holes, the size of the holes has to be reduced to the point where each hole occupies only one cell in the reconfigurable fabric of the chip. This state represents the fragmentation state 2 shown in Figure 3. From this maximum fragmentation state, one can reduce fragmentation in the chip by increasing the sizes of the holes. Eventually, the holes are so large that they may merge together to form a completely empty chip consisting of a single hole occupying the entire reconfigurable fabric of the FPGA chip. In this context, fragmentation is nonexistent (0%) in an empty chip. Based on these semantics, fragmentation becomes irrelevant in the case of a completely full chip and subsequently cannot have a valid value between 0% and 100% in this case. This means that the proposed fragmentation metric can be applied only to a completely or partially empty chip.

• **Fragmentation Factor :** Let a and A be the area of a single empty cell and the entire chip respectively. Let N x N be the number of cells in an FPGA chip. Assume that a hole i consists of k cells. This hole yields a fragmentation factor $f_i = \frac{1}{A}\sum_{j=1}^{k} a = \frac{ka}{N^2 a} = \frac{k}{N^2}$.

• **Fragmentation Metric:** Since the factor $f_i = \frac{k}{N^2}$ gets smaller as many cells are made empty in the chip, it is scaled to reflect maximum fragmentation by subtracting it from 1 as $F = 1 - \left(\prod_i f_i\right)$. F represents the fragmentation metric of the FPGA chip at any moment.

• **Lowest Possible Fragmentation:** An empty chip represents the lowest possible degree of fragmentation. In an empty chip, there is only a single empty area consisting of one hole whose area is N^2a. In this case, $F = 1 - \left(\prod_{i=1}^{1} f_i\right) = 1 - f_1 = 1 - \left(\frac{N^2 a}{N^2 a}\right) = 0$.

• **Highest Possible Fragmentation:** A highly fragmented chip resembles the checkerboard layout shown in Figure 3. Assuming N is even, the number of holes in the chip is $\frac{N^2}{2}$ where each hole occupies a single cell. In this case, $F = 1 - \prod_{i=1}^{\frac{N^2}{2}} f_i = 1 - \prod_{i=1}^{\frac{N^2}{2}} \left(\frac{a}{A}\right) = 1 - \frac{1}{(N)^{N^2}}$.

Although F does not reach exactly 1 as shown, it nevertheless approaches 1 as N gets larger. While this fragmentation metric is similar to the one proposed in [1], its semantics are totally different. Given this formulation of the fragmentation metric, any event that modifies the state of the reconfigurable fabric of the chip can affect the value of F. Events which can do so consist of placing a task on the chip, purging a task from the chip, or moving a task from location to location on the chip. As a result, it is the responsibility of the placement and defragmentation process to constantly update F when these events are witnessed.

2.2 Temporal Management of Defragmentation

While applications are running on an RCS, computation tasks will be queued to be placed on the FPGA chip for acceleration as determined by the

scheduler. At some point, the reconfigurable fabric of the chip is so fragmented that task placement eventually fails. In this case, there is no choice but to perform defragmentation in order to free up room for the waiting tasks. This approach can be called *on-demand defragmentation* since defragmentation is upheld until task placement reaches the failure point. On the other hand, one can opt to perform defragmentation before the failure point. This approach can be called *scheduled defragmentation*. Regardless of which approach is used, tasks are forced to wait in the queue while defragmentation is being performed. Assume that, at any one time, the queue consists of a set of tasks $T = \{t_i: i = 1, 2, \ldots, |T|\}$ where each task t_i requires a time e_i to execute on the FPGA chip, and may wait a given time w_i in T before being placed on the chip. While a task t_i is executing on the FPGA chip, it may be interrupted, so it can be copied to another location if the defragmentation process decides to move it away from other tasks. c_i represents the time it takes to copy task t_i. Hence, one can associate with T a set $E = \{e_i > 0: i = 1, 2, \ldots, |T|\}$, a set $W = \{w_i \geq 0: i = 1, 2, \ldots, |T|\}$, and a set $C = \{c_i \geq 0: i = 1, 2, \ldots, |T|\}$. A task t_i remains in the RCS for a lifetime of $l_i = w_i + e_i + c_i$. When defragmentation is being performed, the waiting times of the queued tasks may increase. This means that w_i for each task will increase while c_i will increase for some tasks, thus increasing the average waiting time $\bar{w} = \frac{1}{|T|}\sum_{i=1}^{|T|} w_i$ and the average copying time $\bar{c} = \frac{1}{|T|}\sum_{i=1}^{|T|} c_i$. Effective temporal management of defragmentation should minimize $\bar{w}$ and subsequently $\bar{l}$. By continually minimizing $\bar{w}$ to ultimately bring it down to zero, system performance is raised back to its initial level before the queue was stalled. If on-demand defragmentation is used, it is possible that, by reaching the placement failure point, the fabric of the FPGA is so severely fragmented that a lengthy defragmentation is needed to release enough fabric areas to service the waiting tasks. This lengthy defragmentation will forcibly increase $\bar{w}$, $\bar{c}$, and subsequently $\bar{l}$. On the other hand, if scheduled defragmentation is used, one can trigger defragmentation at specific intervals to maintain the fragmented state of the FPGA chip constantly below a pre-defined threshold of fragmentation. In this case, although defragmentation is performed repeatedly, it is scheduled to run for short time durations which can benefit the waiting tasks by decreasing $\bar{w}$. The adopted fragmentation metric can be used as a gauge to constantly monitor the fragmented state of the FPGA chip in order to determine the opportune moment to kick-in defragmentation. From this context, it is clear that the metric would not be of any help in the case of on-demand defragmentation since it is triggered only when placement reaches the failure point. There are several ways in which the fragmentation metric can be used to schedule defragmentation. One can monitor the actual value of the fragmentation and decide to schedule fragmentation when this value goes below a defined threshold $F_{threshold}$. This threshold value can be adjusted in real-time based on $\bar{l}$. As $\bar{l}$ increases, $F_{threshold}$ can be increased until $\bar{l}$ is restored to a value that is indicative of a desired performance. On the other hand, one can monitor the rate of change of the metric and use it to schedule defragmentation instead. A rate of change can be defined as $\Delta F = \frac{F_2 - F_1}{t_2 - t_1}$ where F_2 and F_1 are the values of the fragmentation metric at time t_1 and t_2 respectively. It is assumed that event 1 and 2 occur at t_1 and t_2 respectively where $t_1 \leq t_2$. The rate of change can be more informative if it is monitored over a given time window. For instance, the average of the rate of change of the fragmentation metric can be defined as $\overline{\Delta F} = \frac{1}{n}\sum_{i=1}^{n} \Delta F_i$ over a window of n events. A reasonable window would be the time lapse between two successive defragmentations of the chip. As $\overline{\Delta F}$ increases, the fragmentation state of the fabric deteriorates rapidly which should trigger defragmentation. When defragmentation is being performed, $\overline{\Delta F}$ will decrease gradually until $\bar{l}$ reaches a desired value at which point defragmentation can stop. At this point, it is not clear which metric can be effective with what range of values in helping the minimization of $\bar{l}$.

2.3 Spatial Management of Defragmentation

Once defragmentation is started, the next step is to determine where in the fabric of the chip defragmentation ought to be performed. A straightforward approach would be a greedy approach based on reducing the fragmentation metric as much as possible. In that case, one can scan the reconfigurable fabric of the FPGA chip to identify tasks which can help reduce the metric if they are moved to other empty areas on the fabric to be compacted together,. However, when tasks are moved, there may be several empty areas on the chip that can be candidate for hosting these moved tasks. Obviously, the best empty areas are the

ones that contribute most to the reduction of the fragmentation metric.

2.4 Resolution of Defragmentation

When the decision to perform defragmentation is taken, the next step is to determine how much defragmentation is necessary. The amount of defragmentation needed, called *resolution of defragmentation*, can be determined based on the strategy used to schedule defragmentation. If a substantial amount of defragmentation is needed, it is obvious that the waiting times of the tasks currently stalled in the queue will increase. In return, this may decrease the waiting times of other tasks that have yet to enter the queue. On the other hand, if a minimal amount of defragmentation is sufficient, the waiting times of a subset of the tasks currently stalled in the queue will decrease, thus decreasing $\overline{w}$ in the overall. However, in the long run, it may increase the waiting times of other incoming tasks that have not entered the queue yet. In any case, it is not clear to what degree the amount of defragmentation impacts $\overline{w}$ in the short and long run.

2.5 Mechanism of Defragmentation

Essentially, defragmentation consists of moving tasks from one location to another in the FPGA chip. This process requires copying each cell in the rectangular shape of the task from the source location to the destination location. The most straightforward approach can be a three-step procedure in which (i) the computing operations of the task are paused, (ii) the task is relocated block by block, and (iii) the operations of the task are restarted. Pausing the operations of the task will increase its lifetime by $l_i = w_i + e_i + c_i$. In current FPGAs, the time to reconfigure a block, or a cell in a given task, does not depend on its location within the chip. What matters is the number of blocks or cells to be copied. This number, which depends on the size of the task, affects the copying time of the task. In this context, larger tasks will take longer to copy than smaller ones. If the configuration bitstream of each task is kept in an off-chip storage area, copying a task consists of downloading or writing the bitstream to the selected target empty area. Otherwise, the task can be moved from one location to another by reading each cell from within the FPGA chip and writing it to the target location on the chip. Whereas the former approach requires only writing each cell of the task to the target location, the latter approach requires reading each cell from the source location and writing it to the target location. In terms of copying time, the latter approach can almost double the copying time c_i for a given task t_i. There is one exception in which the latter approach can be used to copy a task without interrupting its operations, which is equivalent to reducing $c_i = 0$. A non-trivial procedure was proposed in [9] whereby each CLB in a task, mapped onto a Xilinx Virtex FPGA, is duplicated by copying it to the target area and connecting its inputs and outputs to those of its twin CLB in the source area. This step, which takes one clock cycle, is sufficient for combinational logic. However, in the case of sequential logic, the first step is followed by another step in which the internal state of sequential registers is copied to the newly created CLB. According to the authors, this procedure works for combinational circuits, synchronous free-running clock circuits, synchronous gated clock circuits, and asynchronous circuits. Nevertheless, the authors state that it does not work for special sequential structures such the ones in which lookup tables (LUTs) are configured as distributed RAMs. While this procedure does not interrupt the operation of a task, it requires at least two clock cycles to duplicate a CLB thus increasing the copying time of the task. In return, this prolongs defragmentation time, which by default forces stalled tasks to wait longer in the queue, thus resulting in a net increase of $\overline{w}$.

2.6 Place of Defragmentation in System Architecture

Although defragmentation can be used to sustain application performance in an RCS, it is not clear where between the general purpose processor of the RCS and the FPGA chip it can be integrated. Essentially, one can view the integration of defragmentation in an RCS as a software/hardware co-design problem. In this problem, the management of defragmentation can be implemented (i) completely in a software layer, (i) completely in hardware, or (iii) partially in software and partially in hardware. The first approach requires defragmentation to be integrated in the task management software infrastructure used to support the runtime of the RCS. This approach is labeled **(Place 1)** in Figure 4. However, the second approach requires the implementation of defragmentation to reside on the FPGA board itself. Finally, the third approach requires the partitioning of defragmentation between software and hardware realizations based on formulated constraints in order to achieve specific tradeoffs. In the hardware approach, if the compiled model of defragmentation is sufficiently small, it can reside on a microcontroller, a Complex Programmable Logic Device (CPLD), or an auxiliary FPGA which can be placed on the FPGA board in order to monitor the state of the reconfigurable fabric on the main FPGA chip.

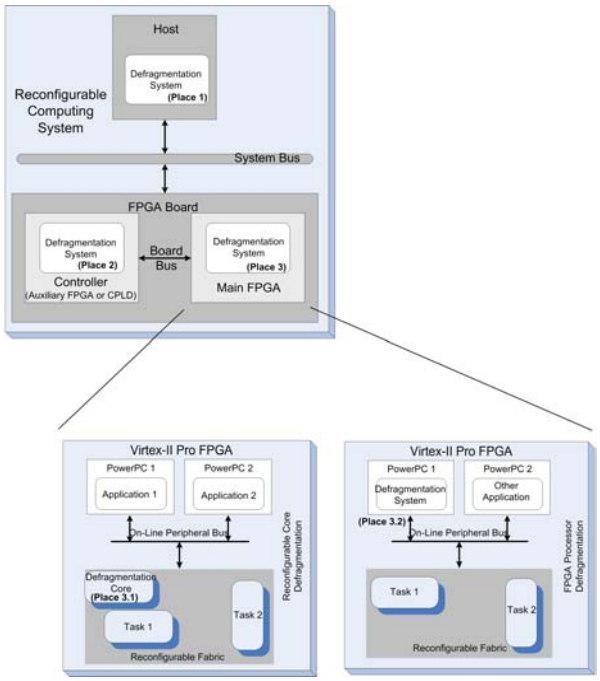

Figure 4. Place of defragmentation in system architecture.

This approach is labeled **(Place 2)** in Figure 4. In this approach, it is assumed that a smaller controller FPGA chip, hosting the defragmentation algorithm, is located beside the main FPGA chip on the FPGA board. Both chips can communicate through direct wires or buss interfaces. Furthermore, if the implementation of defragmentation results in an extremely small compiled model, it can be migrated to reside on the main FPGA itself. This approach, labeled **(Place 3)**, is shown at the bottom of Figure 4 where (i) a defragmentation core can run on the reconfigurable fabric of the FPGA chip, labeled **(Place 3.1)** in Figure 4, or (ii) execute as a compiled executable on the microprocessor of a system-on-chip FPGA such as the Virtex-II Pro FPGA from Xilinx. The latter is labeled **(Place 3.2)** in Figure 4. Most current FPGA vendors offer high-capacity FPGA chips which resemble full-blown reconfigurable system-on-chips (SOCs). These SOCs can contain several processors, a multitude of memory resources, and other fast datapaths.

While in the approach labeled **(Place 3.2)**, the algorithm will consume minimal resources on the reconfigurable fabric, it will occupy a non-trivial area in the approach labeled **(Place 3.1)**, thus resulting in a reduction of the available reconfigurable area for the incoming tasks. However, one can anticipate that communication latencies in the approach labeled **(Place 3.1)** will be significantly lower as compared to the same latencies obtained in the approach labeled **(Place 3.2)**. The latter requires that the SOCs embedded processors communicate with logic mapped onto the reconfigurable fabric through busses such as the Online Peripheral Bus (OPB) offered with the Xilinx Virtex-II Pro. In general, full hardware implementations can provide the fastest response to perform defragmentation on the FPGA chip although at a net cost of increase in the hardware resources present in the RCS. On the other hand, a full software realization produces the slowest responding implementation without increasing the hardware resources in the RCS. Whereas there are obvious advantages and disadvantages to each approach, it is not quite clear at this point which approach can be effective in sustaining performance.

3.0 Experimental Research

Since there are numerous parameters that can impact defragmentation and its capability to sustain performance, it is difficult to model the interactions of these parameters in a coherent mathematical model suitable for tractable analysis. Because of this difficulty, simulative experimentation seems to be the best approach to evaluate defragmentation within the context of system performance. The runtime environment of the RCS can be viewed as a single-queue single-server model in which incoming tasks arrive in a given order determined by the scheduler, and move through the queue before being serviced by the server represented by the FPGA chip in this case. This model can easily be extended to a multiple-queue multiple-server model representing multi-node clustered RCSs supported by multiple FPGA chips. In this

queuing model, the server, which represents the FPGA chip, needs to service the incoming tasks based on a well defined policy. This server has a limited capacity that is represented by the total area of the FPGA chip. As long as some capacity is available in the server, which is equivalent to the availability of empty areas on the FPGA chip, the server continues to service incoming tasks by placing each one on an unoccupied area of the FPGA chip. As a result, there may be more than one task being serviced by the server at any one time. The placement of tasks can be determined by an efficient placement algorithm [1, 4, 6]. If task placement fails, defragmentation can kick-in. After defragmentation is complete, the placement algorithm can resume its role of placing incoming tasks as they arrive. In addition, the queuing model can be instrumented to allow the server to operate in three emulation modes whereby each mode emulates the implementation of defragmentation in a given place in the system architecture described in section 2.6 . Based on various distributions of task sizes, inter-arrival times, and execution times, numerous queuing runs can be performed in order to observe how model factors affect model responses. In this model, the factors are the fragmentation threshold, resolution of defragmentation, task arrival, execution, and copying time distributions, whereas the responses can be any of the following parameters: (i) task statistics such as *average waiting/copying times or lifetimes of all tasks* and *maximum waiting time or lifetime of all tasks*; (ii) queue statistics such as *average queue length* and *maximum queue length*; (iii) and server statistics such as *total idle time*, *total busy time*, and *average utilization*. By collecting these statistics, meaningful relationships between the model factors and responses can be uncovered. Among the possible functional relationship between factors and responses, the average task waiting time can be evaluated in relation to fragmentation threshold, defragmentation resolution, task arrival rate, average task execution time, and average task size, meaning average task waiting time = f_1 (fragmentation threshold, defragmentation resolution, task arrival rate, average task execution time, and average task size). Similar functional evaluations can be applied to the average task lifetime (f_2), average sever utilization (f_3), average number of tasks in the system (f_4), or maximum queue length (f_5).

The analysis of these relationships can be used to interpret the interplay between the factors and responses of the model. For example, obtaining the function defined in f_1, and f_2 shows how defragmentation affects the way tasks spent their time in the RCS before they die away. The function defined in f_3 shows how defragmentation affects how full or empty the FPGA chip throughout the execution of the applications in the RCS. The function defined in f_4 shows how defragmentation affects the size of the task population that the RCS can support. Finally, the function defined in f_5 shows how defragmentation affects the size of the buffer in which the tasks are stored before being downloaded to the FPGA chip. Additional functions can be formulated in order to understand the interplay between the same factors and other model responses.

4.0 Conclusion

In this paper, the problem of defragmentation of computation tasks on FPGA chips embedded within RCSs has been presented. The paper proposes a methodology to study the impact of defragmentation on performance sustainment by providing tentative answers to a set of six questions. This methodology is based on extensive simulations in which various factors of defragmentation can be modulated in order to tally their impact on a set of defined system responses. A thorough understanding of this impact can help in formulating a broad framework in which task scheduling, placement, and defragmentation are accurately coupled in RCSs with the objective of sustaining maximum application performance.

5.0 References

[1] J. Tabero, J. Septien, H. Mecha, D. Mozos, and S. Roman, "Efficient Hardware Multitasking through Space Multiplexing in 2D RTR FPGAs," *Euromicro Digital System Design Conference*, September 2003, pp.

[2] G. Wigley and D. Kearney, "The management of Applications for Reconfigurable Computing Using an Operating System," *The 7th Asia-Pacific Conference on Computer System Architecture*, 2002, pp. 73-81.

[3] O. Diessel and G. Wigley, "Opportunities for Operating Systems Research in Reconfigurable Computing," University of South Australia ACRC-99-018, Aug. 1999, available at http://citeseer.ist.psu.edu/diessel99opportunities.html.

[4] H. Walder, C. Steiger, and M. Platzner, "Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-Hashing," *International Parallel and Distributed Processing Symposium*, April 2003, pp.

[5] H. Walder and M. Platzner, "Non-preemptive Multitasking on FPGAs: Task Placement and Footprint Transform," *The 2nd International Conference on Engineering of Reconfigurable Systems and Architectures*, June 2002, pp. 24-30.

[6] M. Handa and R. Vemuri, "An Efficient Algorithm for Finding Empty Space for Online FPGA Placement," *Design Automation Conference*, San Diego, CA, June 2004, pp. 960-965.

[7] H. Walder and M. Platzner, "Online Scheduling for Blockpartitioned Reconfigurable Devices," *Design, Automation and Test in Europe*, Mar. 2003, pp. 290-295.

[8] M. Handa and R. Vemuri, "Area Fragmentation in Reconfigurable Operating Systems," *Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, NV, June 2004, pp.

[9] M. G. Gericota, G. R. Alves, M. L. Silva, and J. M. Ferreira, "Run-Time Management of Logic Resources on Reconfigurable Systems," *Design Automation and Test in Europe*, March 2003, pp. 974-979.