Evaluation of FPGA Partitioning Schemes for Time and Space Sharing of Heterogeneous Tasks

Umar Ibrahim Minhas, Roger Woods, and Georgios Karakonstantis

Queen's University Belfast

Abstract. Whilst FPGAs have been integrated in cloud ecosystems, strict constraints for mapping hardware to spatially diverse distribution of heterogeneous resources at run-time, makes their utilization for shared multi tasking challenging. This work aims at analyzing the effects of such constraints on the achievable compute density, i.e the efficiency in utilization of available compute resources. A hypothesis is proposed and uses static off-line partitioning and mapping of heterogeneous tasks to improve space sharing on FPGA. The hypothetical approach allows the FPGA resource to be treated as a service from higher level and supports multi-task processing, without the need for low level infrastructure support. To evaluate the effects of existing constraints on our hypothesis, we implement a relatively comprehensive suite of ten real high performance computing tasks and produce multiple bitstreams per task for fair evaluation of the various schemes. We then evaluate and compare our proposed partitioning scheme to previous work in terms of achieved system throughput. The simulated results for large queues of mixed intensity (compute and memory) tasks show that the proposed approach can provide higher than 3× system speedup. The execution on the Nallatech 385 FPGA card for selected cases suggest that our approach can provide on average 2.9× and 2.3× higher system throughput for compute and mixed intensity tasks while 0.2× lower for memory intensive tasks.

Keywords: Cloud Environments · Data Centers · Space Sharing

1 Introduction

Cloud computing offers users ubiquitous access to a shared pool of resources, through centralized data centres. With increasing device sizes and efficiency for high performance computing, there has been an increased interest in recent times to integrate Field Programmable Gate Arrays (FPGAs) in data centres [5][11]. However, their architecture and programming environment presents a different resource sharing model when compared to software programmable accelerators.

The challenge lies in sharing the device space by accommodating multiple heterogeneous tasks at one instance of time. Heterogeneous tasks in our context are defined by heterogeneity in resource utilization (compute, memory, logic) and execution time. Optimization of system's resource utilization in time and space when executing these tasks in a shared environment is a challenging task, leading to suboptimal compute density and system throughput.

In software-based systems, a runtime approach can map a task to any portion of underlying hardware. This together with *microsecond* latency context switching between tasks, provides flexible sharing of resources. For FPGAs, the tasks are custom designed and mapped spatially on the device off-line. This, along with the reconfiguration overhead associated with task initiation, places extra constraints for efficient utilization of FPGA resources [9].

A common way of sharing the FPGA space is to partition it into partially reconfigurable regions (PRRs) which can be configured independently in time and partially in space. Flexibility in space is partial as incoming tasks can only be placed in one of the statically defined PRR via dynamic partial reconfiguration (DPR) at runtime. This means that tasks with diverse resource needs are mapped to the same homogeneous PRR which may result in inefficient resource utilization. To address this challenge, researchers have looked at providing more flexibility in space using heterogeneous PRRs and multiple bitstreams for a single task [3]. Although this approach increases the system throughput via intelligent off-line and runtime PRR design, the same intrinsic idea of mapping more than a single task to the same PRR still may lead to inefficient resource utilization.

This work first analyses the effect on compute density due to constraints imposed by PRR. To achieve this, we create multiple bitstreams per tasks for ten real high performance computing (HPC) tasks, for domains such as graph analytics, dense linear algebra, scientific computing, etc., allowing exposure of the area-throughput trade-off. This design space exploration (DSE) allows us to estimate the average utilization of heterogeneous resources (Logic Cells, DSPs, BRAMs) in a homogeneous PRR. Furthermore, along with the help of an exhaustive simulator, the DSE allows us to gauge the effect on system *speedup* for various PRR optimizations of runtime scheduling using real HPC tasks.

Secondly, we also evaluate an alternative approach to PRR by hypothesizing that a higher compute density can be achieved via static partitioning and mapping (SPM) of heterogeneous bitstreams. The SPM looks to provide complete spatial independence as heterogeneous tasks can be mapped to custom designed regions utilizing all of the resources on the FPGA. This only provides partial time independence, however, as tasks sharing the FPGA need to be reconfigured and executed at the same time, resulting in stalling by the longest running task.

Thirdly, our work compares both approaches while varying system design parameters. To achieve this, the simulator allows scheduling of large task queues with varying execution times to estimate average system *speedup*. Moreover, implementation of selected cases on an actual FPGA allows us to analyze the constraints of both approaches when targeting compute or memory intensive tasks, and report on performance in terms of System Throughput (STP), a metric defined specifically for multi-task workloads processing. The above mentioned DSE and PRR optimisations enable a fair comparison of both approaches.

The results show that SPM can provide a higher compute density while allowing for bitstreams generation from a higher level Open Computing Language (OpenCL) representation. SPM can be complemented with data center workload characterization [1] to select the best approach for varying environments. Statically

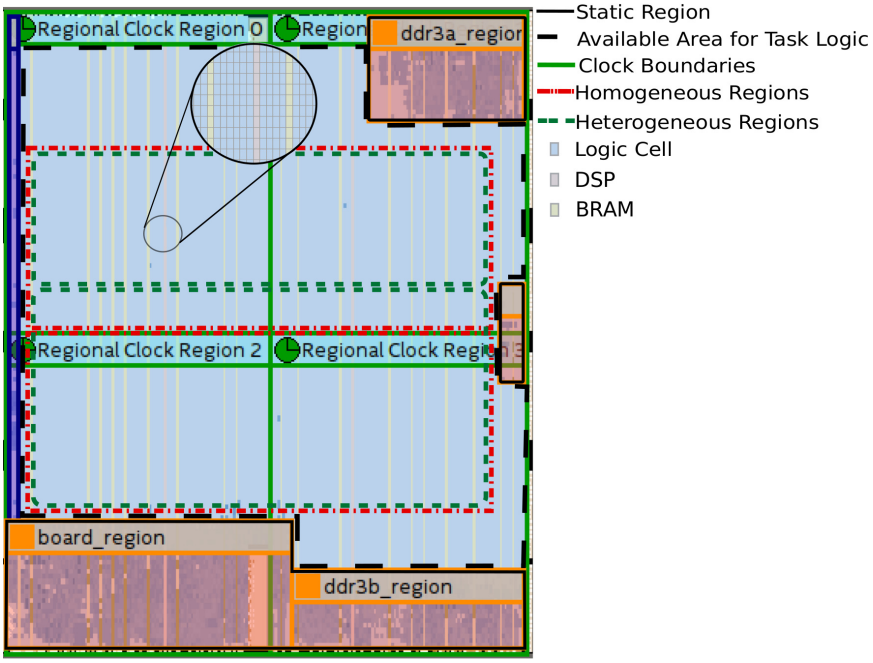

Fig. 1. FPGA Partitioning for PRR

generated high compute density bitstreams fit well with the idea of providing users with a library of optimized IPs allowing tasks to access the FPGA resource as a service (Amazon Marketplace for Amazon FPGA Image).

We first discuss the motivation of this work based on previous studies in Section 2. We then present our implementation and evaluation methodology in Section 3 and our detailed experimental evaluation in Section 4 followed by conclusions in Section 5.

2 Background and Motivation

Cloud services are being used by range of users with diverse computing requirements which vary with task size and type [16]. In FPGA, the compute versus memory intensity of the task, suggests the need for FPGA sharing by heterogeneous tasks in order to achieve maximum system utilization. For sharing, the FPGA is partitioned into rectangular PRRs which are configured typically with a new bitstream via DPR, independently of the processing going on in other PRRs [17]. This provides independence in time to each PRR, such that a task A running in a PRR can be instantly replaced by task B, when task A finishes.

The design of PRRs is challenging since the spatial distribution of various types of resources on FPGA is not uniform or homogeneous (Fig. 1). The whole FPGA can be represented as a matrix, with dimensions $X \times Y$, of tiles where each

tile, $x_i y_j$, represents a resource type; logic cell, DSP block or BRAM. Resources of same type form the *columns*, y_{dsp}, y_{bram}, etc., of a matrix where each row, x_i, contains at least 1 tile involving all type of resources. Furthermore, the FPGA is divided into multiple clock regions across both the vertical and horizontal axes, where the crossing of the region boundary requires custom logic implementation.

Now since the tasks are physically mapped to this diverse distribution of resources, their relocation at runtime is challenging, particularly along the horizontal axis [6]. For more complex mappings, modern bitstream relocation techniques [14] allow for relocation from one region to another vertically only, due to the column-based FPGA architecture, whilst permitting routing of interface connections and clock. However, such a relocation scheme cannot happen from a non-clock crossing region to a clock crossing region and vice versa. Thus, relocation is only possible among homogeneous regions along the y-axis and at discrete starting points with a step size equal to height of clock regions (Fig. 1), in line with work on partially reconfigurable systems for independent tasks [17].

These mapping constraints require PRRs to be majorly homogeneous and along the y-axis which may lead to inefficiency in resource utilization by heterogeneous tasks. Firstly, after omission of the static area used for memory interconnects near I/O pins and other hard static logic, the homogeneous region along the y-axis can be as low as 60% area of the FPGA [17]. The concept is explained in Fig. 1 where the marked boundaries represent the total available area and area distribution for homogeneous PRRs and heterogeneous PRRs (discussed in Section 3.2) after considering static resources and clock regions. In this case, PRR area is limited to 80 *rows* of resources compared to total 128 *rows* of FPGA along the vertical axis, with further limitations on horizontal axis. Secondly, within the rectangular boundaries defined for any task, the actual area being allocated to task may be lower than the area available in that region, namely 38% - 51% [18] which is similar to our own implementation of HPC tasks (Section 4). This is worsened in case of fixed PRRs due to diverse spatial placement of different types of resources.

Whilst mapping optimizations using PRRs is well researched [3][17], for the first time, this work intends to analyze the effect on compute density caused by the constraints of PRRs and inefficient utilization of resources when mapping heterogeneous tasks. Firstly, we create a large design space using a range of real high performance computing tasks while exposing the area-throughput trade-off, using the biggest selection of the most relevant HPC tasks to date [4][15]. This allows us to quantify the heterogeneity in resource utilization by modern workloads when mapping to FPGA and to project the need for heterogeneous mapping. The DSE also allows us to quantify various existing PRR optimizations in literature using a range of real workloads.

We then propose SPM of tasks in heterogeneous regions as a mean to achieve higher compute density. Although the technology has supported this approach, this is the first time it has been analyzed from a high level perspective for use in space sharing FPGAs in data centers. The approach aims to provide complete spatial independence for highly optimized mapping on account of partial time

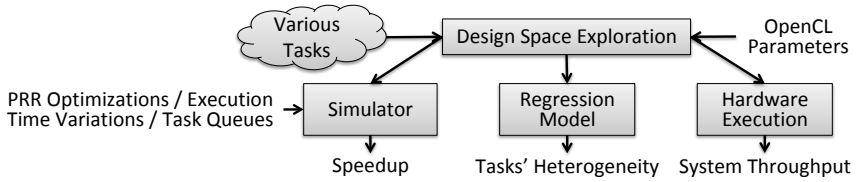

Fig. 2. Summary of Implementation and Evaluation Methodology

independence. Time independence is partial as all tasks need to be reconfigured at the same time. We aim to quantify this and comment on design parameters that affect system performance. Finally, we fairly compare both approaches using the DSE on a flexible simulator as well as real hardware execution measurements.

3 Implementation and Evaluation Methodology

In this section, we describe multiple aspects of the design environment (Fig. 2). In particular, we start with multiple bitstreams generation for each task and then define the optimizations applied to PRR mapping. Finally, we briefly define our simulator and metrics used for evaluation.

3.1 Multiple Tasks' Bitstreams with Area-Throughput Trade-off

A key goal is to generate multiple hardware bitstreams of the same task that provide a *speedup* corresponding to the resources used. Using an area-throughput curve, this allows for precise quantification of the variation in compute density with resource utilization due to different partitioning strategies.

To achieve this, we make use of the OpenCL framework for heterogeneous parallel programming that both provides abstraction of parallelism and a high level DSE model for tuning the underlying hardware mapping. In addition to OpenCL, we use general high level synthesis parameters, to scale the task over multiple parallel compute units *(CUs)*; multiple pipelines can be defined via a Single Instruction Multiple Data *(SIMD)* parameter, whilst the key compute intensive loops can be unrolled via the UNROLL *(U)* parameter. For some tasks, we vary task-specific parameters such as block size or number of rows, where these define the parallel processing of a defined parameter size. All these parameters allow scaling of the underlying hardware by varying the number of custom parallel data paths for each task.

3.2 Partially Reconfigurable Regions Mapping Optimizations

Using the same created design space, we can quantify the various optimizations presented in earlier studies and which are important to fairly compare PRR with SPM. The optimizations are mainly targeted at avoiding segmentation, causing vacant regions of FPGA, by varying the sizes of used bitstreams. Basic PRR

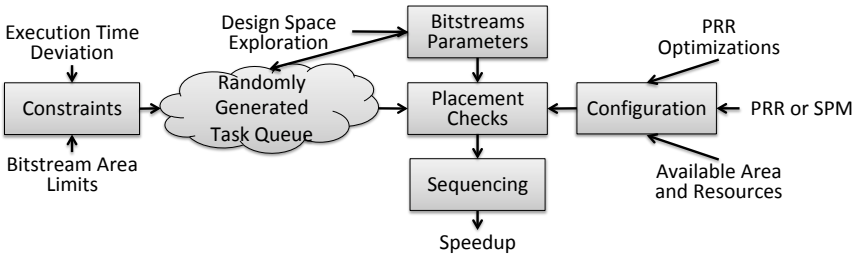

Fig. 3. Runtime Simulator

mapping generates homogeneous regions as well as a single bitstream for each task corresponding to that region. Among the optimizations, the first one is called Elastic resource allocation which looks at adjacent PRR regions. If they are free, the approach attempts to fit larger bitstreams of the same task in this combined region, thus replacing the current task bitstream with a larger one to gain a *speedup* [17].

Another way to increase mapping flexibility is to partition the FPGA into heterogeneous PRRs which offer different number of resources[3] [5]. The tasks are then custom designed for one of the PRRs. Heterogeneous PRRs can be defined by including a different ratio of each heterogeneous resource type. However, in our case, the device size is not big enough to benefit from such an approach, so we define heterogeneous PRRs by varying the number of each type of resources while their relative ratios remain the same (Fig. 1). We define the areas on top of homogeneous regions which means it can either be configured as homogeneous or heterogeneous, allowing flexibility in mapping options from the generated design space. Another optimization that is made possible by heterogeneous PRRs is using smaller (contracted) bitstreams for tasks when none of the original bitstream can be fit into a region [3].

Finally, we provide simulated results for continuous y-axis, i.e. the hypothetical performance gains that can be made if the bitstream relocation step size is reduced to a single *row* by future technology support. At present, this can be achieved by generating multiple bitstreams, equal to the number of *rows* within each clock region, by varying starting y-coordinates for each unique bitstream.

3.3 Runtime Simulator and Hardware Implementation

The key configurable parameters and functional blocks of the runtime simulator are summarized in Fig. 3. The DSE provides the bitstreams' characteristics such as area and resource utilization, and relative speedup. This along with tasks' parameters, such as their execution time and limitation on resource utilization of bitstreams as per the available resources, is used to generate a task queue which is then fed into task placement and sequencing modules. For evaluation of average *speedup*, these modules then treat the task queue mapping as a rectangle fitting problem with configurations set for continuous y, homogeneous

and heterogeneous PRRs along with other mapping optimizations or use SPM, while keeping under the overall available resources. As for the sequencing of tasks, we use a basic first fit heuristic which takes the task queue and tries to fit tasks in incoming sequence.

For evaluation of compute density on hardware, the bitstreams for SPM have been generated using Intel OpenCL SDK for FPGA, and all tasks run at the same frequency. This, however, is not a limitation of design as varying frequencies can be used for statically generated task cores. Also for PRR evaluation, OpenCL is used to generate the intermediate design files and constraints file, then modified to include bounding rectangular regions, as per defined PRRs for logic placement before generating the final bitstream. Similar efforts were used to map the largest possible bitstream configurations both for PRR and SPM within their respective area constraints.

3.4 Metrics

Assessing the system performance of a multi-task workload running in parallel on a single hardware is challenging as the performance of individual tasks may not entirely relate to system performance. We use two different metrics for simulated and hardware results to have a comprehensive assessment. First, simulation of large task queues allows us to project average *speedup*, measured as variation in execution time of complete queue. Secondly, for measuring compute density, we use various configurations of bitstreams of same tasks implemented on hardware which consume varying execution times to process the same data size, as well as allow to share FPGA with different number of tasks. We then use the STP metric [7] as defined by:

$$STP = \sum_{i=1}^{n} NP_i = \sum_{i=1}^{n} \frac{C_i^{SP}}{C_i^{MP}} \tag{1}$$

where NP is each task's normalized progress defined by the number of clock cycles it takes in single (C_i^{SP}) task mode when the task has all of the resources of FPGA, compared to multi (C_i^{MP}) task mode, when it is sharing the space with other tasks. Here, n defines the number of tasks sharing the FPGA.

3.5 Evaluated Tasks

We have considered a number of tasks belonging to various computing dwarfs [2] and application domains with varying ratio of compute and memory operations.

a) Page Rank (PR) is a graph analysis algorithm used for link analysis of web pages, social networks, etc [13].

b) Alternative Least Squares (ALS) based Collaborative Filtering is a verified approach based on the aggregated behavior of large number of users used to develop recommender systems for commercial domains such as Netflix [19].

c) Lower Upper Decomposition (LUD) is an important dense linear algebra used for solving systems of linear equations with reduced complexity [4].

Table 1. Use Cases Characteristics where the step size is 2×, unless otherwise specified.

Use Case	Dwarf	Data Size	Bitstreams Scaling	Speedup
PR	Sparse Linear Algebra	Pages: 64K	(CU: 1,2,4) × (U: 1, 2, 4)	6×
ALS	Sparse Linear Algebra	Users: 4K	(CU: 1, 4)×(U: 1, 4)	2×
BOP	Structured Grids	Options: 2K	CU1 × (U1, U2, U4, U8, U16); CU2×U16; (CU: 3, 4, 5)×U8	21×
BFS	Graph Traversal	Nodes: 64K	U: 1 - 16	5×
SpMV	Sparse Linear Algebra	X×Y: $4K \times 4K$	U: 1 - 32	190×
FDTD	Structured Grids	X×Y×Z: 512 × 512×1K	Block Size: 1 - 16	13×
LUD	Dense Linear Algebra	X×Y: 4K×4K	CU1×(U: 1 - 16); (CU: 2, 3)×U16	18×
VD	Structured Grids	Resolution: 4K	Parallel Rows: 1 - 32	8×
SGEMM	Dense Linear Algebra	X×Y: 4K×4K	SIMD1×(U: 1 - 64); SIMD4×(U: 32 - 64)	204×
NW	Dynamic Programming	X&Y: 4K	Block Size: 2 - 128	33×

d) Binomial Option Pricing (BOP) is a key model in finance that offers a generalized method for future option contract evaluation and for options with complex features [12].

e) Breadth First Search (BFS) is a challenging and important graph traversal algorithm forming the basis of many graph-processing workloads [4].

f) 3 Dimensional Finite Difference Time Domain (FDTD) is an important numerical method for electromagnetic propagation modeling in space [10].

g) Sparse Matrix Vector Multiplication (SpMV) is an important sparse linear algebra algorithm used in scientific applications and graph analytics, etc [8].

h) Matrix Matrix Multiply (SGEMM) is used in various compute intensive algorithms and benchmarks [8].

i) Video Downscaling (VD) is used by a range of media streaming services for real-time bandwidth reductions [10].

j) Needleman-Wunsch (NW) is a bioinformatics optimization algorithm used for protein sequence alignment [4].

3.6 System Hardware

The DSE has been accomplished via Intel OpenCL SDK for FPGAs v 16.1, performed on a Nallatech 385 with an Intel Stratix V GX A7 FPGA chip and 8GB DDR3 memory. The A7 chip has 234,720 ALMs, 256 DSPs as well as 2,560 M20K BRAM blocks. The runtime simulations are performed via Python v3.3.7.

4 Results And Analysis

Our implementation of real tasks is given in Table 1. Besides providing real area numbers for spatial mapping problem, the implementation provides area-throughput graphs for HPC tasks. To measure the *speedup*, the baseline throughput,

Table 2. Average Resource Utilization when Using PRRs

Resource	Custom Regions		Homogeneous PRRs	
	Avg. Util.	Min. / Max. Util.	Avg. Util.	Min / Max Util.
Logic	52.54%	30.36% / 79.40%	37.12%	18.47% / 61.19%
Block RAM	60.56%	15.49% / 95.82%	45.05%	10.07% / 91.91%
DSPs	32.33%	0.0% / 97.0%	26.30%	0.0% / 97.0%

corresponding to the lowest area bitstream, is defined by the serial pipelined benchmark implementation. The maximum throughput is defined by the largest bitstream, limited by FPGA resources. We have generated $4 - 9$ bitstreams per task providing $2 - 204\times$ maximum *speedup* compared to slowest bitstream with *speedup* for each task mentioned in Table 1. The table also mentions the parallelization used for each task, such as number of compute units, unrolling of main computing loop, using SIMD pragma for work items parallelism and data block size variation where elements in a block are executed in parallel and relate to resources utilized in mapping. The generation of multiple bitstreams is a key step in evaluating the mapping strategies as we discuss in coming sections.

4.1 Analysis of Heterogeneous Tasks

Using the DSE, we analyse the heterogeneity in resource utilization by tasks. We mainly focus on three resources, Logic cells, DSPs and BRAMs and evaluate the inefficiency in resource utilization caused by the rectangular and fixed size shapes of PRRs resulting in homogeneous regions. We present percentage utilization of resources from two perspectives.

The first case calculates percentage resource utilization compared to the bounding box where dimensions are custom defined for each bitstream, as per bitstream's resource requirements. We use all of the bitstreams which are smaller than the largest PRR. The second case deals with percentage utilization compared to the PRRs available on the FPGA. We use 4 sizes of heterogeneous PRRs (Fig. 1).

In total, there are 80 *rows* of FPGA that can be configured as a single region (PRR-1) or a set of two homogeneous regions of 40 *rows* each (PRR-2). We define two more heterogeneous PRRs, namely 30 (PRR-3) and 50 (PRR-4) *rows*, based on the sizes of generated bitstreams. Note that either the homogeneous or heterogeneous PRRs can be used at a single instance of time.

We select bitstreams for each task that would maximize the resource utilization in each of 4 PRRs, i.e. up to 4 bitstreams per task and give average percentage resource utilization by these bitstreams compared to their respective PRRs. The measurements in Table 2 show that due to the homogeneous nature of PRRs, the logic, DSP and BRAM utilization is limited to 37%, 26% and 45% on average.

4.2 Runtime Simulation

In this section, we use our simulator to analyse various mapping strategies. Firstly, we examine the *speedup* achieved by various improvements on the PRR mapping, as explained in Section 3.2. We use three different mapping strategies, namely the continuous y-axis, heterogeneous PRRs and homogeneous PRRs and their respective bitstreams (Fig. 1). Please note that this is a study of resource utilization efficiency of various mapping approaches and does not consider data transfers from host CPU memory to DRAM memory on the FPGA board. Furthermore, the DRAM to FPGA on-chip memory transfers are not considered as bottlenecks for simulation purposes, but their effect is discussed in more detail in next section using real execution on hardware.

The runtime scheduler performs Elastic and Contract optimizations, as explained in Section 3.2. We use the actual measured relative throughput of various bitstreams of tasks to calculate the new execution time of tasks. For Contract, we found out that if the difference in *speedup* for a smaller bitstream replacing the bigger is too large, the total execution time increases rather than decrease. Thus, we limited the allowed *speedup* degradation for smaller bitstreams to 5×.

The graphs in the Fig. 4 show the *speedup* achieved for various configurations. Generally, Elastic is more useful with gains up to 1.24× whereas the best gain for Contract is 1.05×. Optimizations benefit more on heterogeneous mapping to tackle segmentation, hence, the gain is negligible for our case of only two heterogeneous regions while no gain is achieved for homogeneous regions.

Next we investigate gains made by SPM in comparison to PRR. For the SPM, we either use the same region as used for PRR (Homogeneous Regions in Fig. 1) and call it Partial Static or use all of the available area for task logic (Fig. 1) and call it Whole Static. This approach helps to differentiate between the *speedup* achieved by heterogeneous mapping in the same region as well as the gains made by the availability of extra logic when mapping statically.

As the results in Fig. 5 show, a key finding is that SPM gives on average 4.6× higher throughput, measured in terms of total execution time for a set size of tasks queue. A 2.4× *speedup* is achieved via heterogeneous mapping while the rest is achieved via use of higher resource availability. The results show that if the y-axis can be made continuous, then a throughput gain of 2× can be achieved.

So far, the reported *speedup* numbers have considered an ideal scenario for SPM by considering all of the tasks sharing the FPGA at any time, have same execution time. This is not the case for real workloads. Next, we vary the execution time of tasks and report on *speedup* achieved. We use a uniform distribution for execution time and vary the range of distribution.

The results shown in Fig. 6 depict a surprising trend. Even with increasing range of execution time by up to 32× (beyond this range a reconfiguration overhead would become negligible for most tasks), the *speedup* decreases but remains higher than 3×. This is because on average, the device under test may be used by 3 or less tasks using SPM, as constrained by the size of FPGA and tasks bitstreams. Thus, a task may stall up to two tasks or a maximum of about 50% resources with an average much lower than that. Stalls by smaller tasks

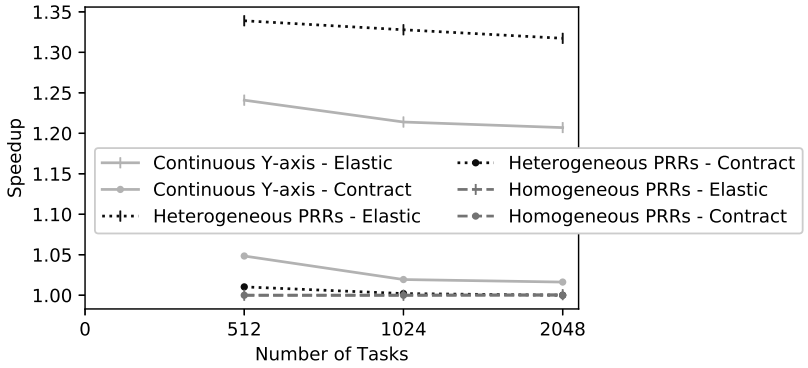

Fig. 4. *Speedup* achieved by Optimization of PRR mapping on various bitstreams

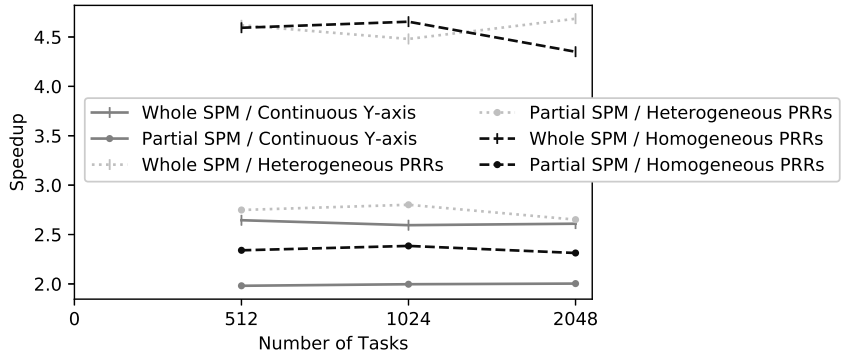

Fig. 5. *Speedup* achieved by SPM versus PRR Mapping

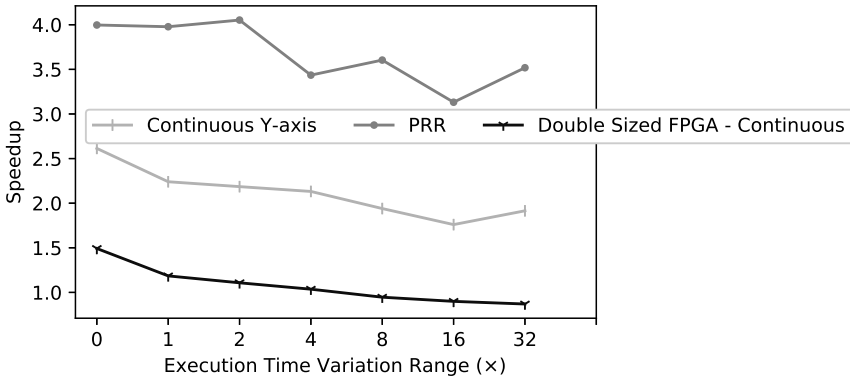

Fig. 6. *Speedup* Variation of SPM with Variation in Execution Times (Tasks = 1024)

are overcome by the higher average compute density and gains made when the longest running task is not the smallest. However, to gauge the effect of the

approach on bigger devices, we estimate the *speedup* possible by increasing the size of available resources while keeping the size of tasks' bitstream the same. The results show that for a device double the size, the gains drop below 1 for an execution time variation greater than 4×. Such a study can help optimize the number of tasks that may be shared at a single instance of time.

4.3 Evaluation on Hardware

A key limitation of using SPM is the need to generate each multi-task heterogeneous bitstream separately. This limitation can be overcome partially by benchmarking cloud and data center workloads to estimate the frequency and data sizes of incoming tasks [1]. This can help decide the combination of tasks that may be shared on a single FPGA as well as the percentage resource allocation for each task. These decisions, apart from helping with a higher resource utilization on-chip, result in minimizing bottlenecks in off-chip resources, such as DRAM access. To analyze this further, as well as provide numbers for throughput for real hardware execution, we discuss some of the extreme cases below using the STP metric defined in Section 3.4.

In terms of resource utilization, SPM resulted on average 60%, 59% and 129% higher logic, BRAM and DSP utilization compared to PRR. Furthermore, Figs. 7, 8 and 9 show the achieved STP for PRR and SPM for two compute, memory and mixed (one compute/one memory) intensive tasks, respectively. The graphs also show the difference in factor between the execution times of both tasks on the second y-axis. In our experiments, the execution time between both tasks varied by up to 5108×, 14× and 361× for compute intensive, memory intensive and mixed tasks. Note that for SPM, NP for each task is calculated using the time for longest running task.

STP for PRR stays relatively uniform with variation in data sizes while for SPM it generally reduces with increase in difference of individual execution times. The results show that for compute intensive and mixed tasks, SPM performs on average 2.9× and 2.3× better than the PRR mapping, respectively.

For memory intensive tasks, the increase in resource utilization did not result in a performance increase. This is because the for memory intensive tasks, the increase in throughput via higher utilization of on-chip compute resources is limited by external memory access latency and bandwidth. For memory intensive tasks, PRR has 1.25× higher STP on average than the SPM.

Finally, for all cases, the trend for SPM is not entirely dependent on the variation in execution time of tasks sharing the FPGA. This is because it also depends on the percentage resource utilization as well as the NP of the longest running task. To explain this further, we present another set of results where we have 4 tasks sharing the FPGA. However, we focus on a single task, LUD, and use two different SPM configurations. In SPM 1, LUD has a minimum number of resources while in SPM 2, it is allocated more such that it has a 10× higher individual NP in SPM 1 compared to SPM 2. Furthermore, we select data sizes for the rest of the tasks such that their execution time is similar to each other. We then vary the data size of LUD (size of square matrices from 128 to 1024 in

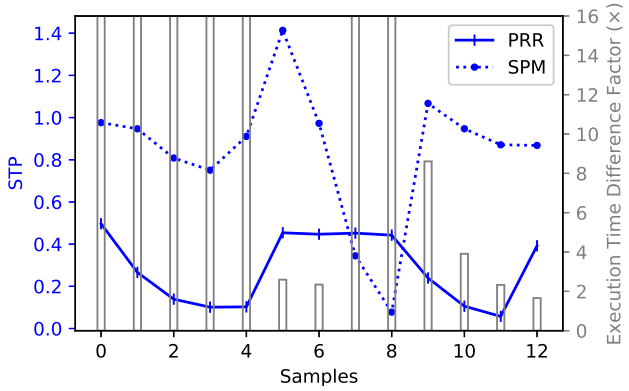

Fig. 7. *STP* Variation with Data Sizes for 2 Compute Intensive Tasks - MM and LUD

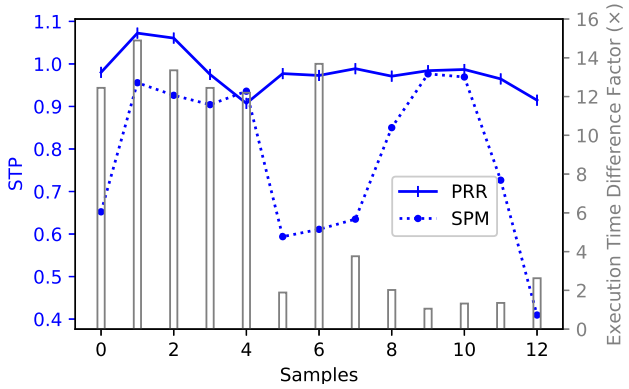

Fig. 8. *STP* Variation with Data Sizes for 2 Memory Intensive Tasks - ALS and PR

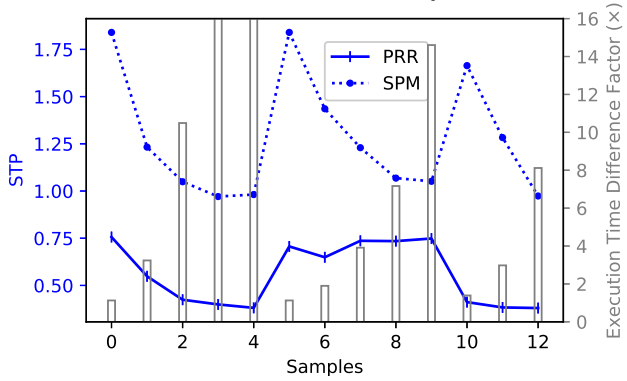

Fig. 9. *STP* Variation with Data Sizes for One Compute and One Memory Intensive Tasks - LUD and PR

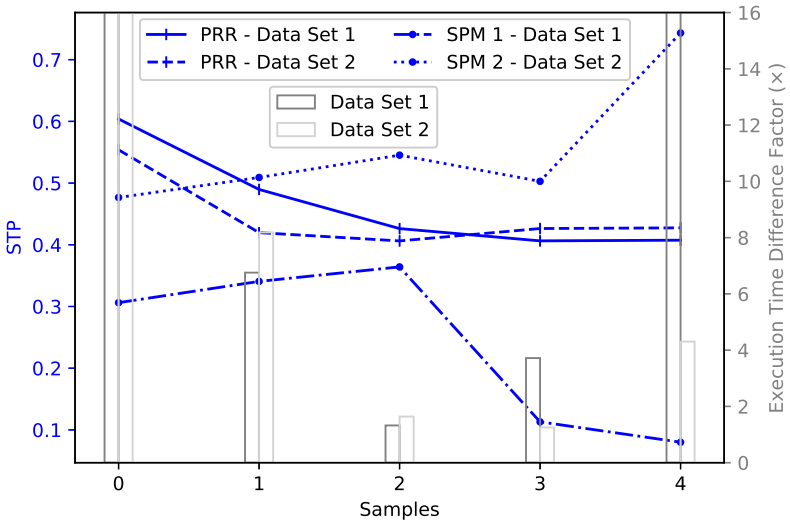

Fig. 10. *STP* Variation for PRRs and Static Designs for Two Configurations using Four Tasks - SPMV, NW, LUD, PR

steps of 2×). The resulting *STP* presented in Fig. 10 shows that for the SPM 1, the PRR performs 1.9× better than the SPM while for SPM 2, the SPM performs 1.2× better than PRR for the same data sizes of LUD. Also even for the second case, SPM performs worse for first sample projecting that sharing 4 tasks on this size of an FPGA reduces the average system throughput.

5 Conclusion

This work analyses the constraints of mapping bistreams of heterogeneous tasks to FPGA at runtime and their effect on compute density when using partially reconfigurable regions for space shared multi-task processing. Static partitioning and mapping of tasks to achieve higher *speedup* and system throughput is proposed and several aspects of each approach are evaluated via design space exploration using a range of HPC tasks, a comprehensive simulator and evaluation on hardware. Static partitioning provides up to 2.9× higher system throughput and facilitates a completely software based implementation of a multi-task computing environment without requiring low level support for PRR.

Acknowledgment

The work was supported by the European Commission under European Horizon 2020 Programme, grant number 6876281 (VINEYARD).

References

1. Abdul-Rahman, O.A., Aida, K.: Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon. In: International Conference on Cloud Computing Technology and Science. IEEE (2014)
2. Asanovic, K., et al.: The landscape of parallel computing research: A view from Berkeley. Tech. rep., Technical Report UCB/EECS-2006-183, EECS Department, University of Berkeley (2006)
3. Charitopoulos, G., Koidis, I., Papadimitriou, K., Pnevmatikatos, D.: Run-time management of systems with partially reconfigurable FPGAs. Integration, the VLSI Journal **57** (2017)
4. Che, S., et al.: Rodinia: A benchmark suite for heterogeneous computing. In: International Symposium on Workload Characterization. Ieee (2009)
5. Chen, F., et al.: Enabling FPGAs in the cloud. In: Conference on Computing Frontiers. ACM (2014)
6. Enemali, G., Adetomi, A., Seetharaman, G., Arslan, T.: A functionality-based runtime relocation system for circuits on heterogeneous FPGAs. IEEE Transactions on Circuits and Systems II: Express Briefs **65**(5) (2018)
7. Eyerman, S., Eeckhout, L.: System-level performance metrics for multiprogram workloads. IEEE Micro **28**(3) (2008)
8. Gautier, Q., et al.: An OpenCL FPGA benchmark suite. In: 2016 International Conference on Field-Programmable Technology. IEEE (2016)
9. Huang, M., et al.: Programming and runtime support to blaze FPGA accelerator deployment at datacenter scale. In: Symposium on Cloud Computing. ACM (2016)
10. Intel: Developer zone. Intel FPGA SDK for OpenCL. https://www.intel.com (2018)
11. Minhas, U., et al.: Nanostreams: A microserver architecture for real-time analytics on fast data streams. IEEE Transactions on Multi-Scale Computing Systems (2017)
12. Minhas, U.I., Woods, R.F., Karakonstantis, G.: Exploring functional acceleration of OpenCL on FPGAs and GPUs through platform-independent optimizations. In: International Symposium on Applied Reconfigurable Computing. pp. 551–563 (2018)
13. Page, L., et al.: The pagerank citation ranking: Bringing order to the web (1998)
14. Pham, K.D., Horta, E., Koch, D.: Bitman: A tool and API for FPGA bitstream manipulations. In: Design, Automation & Test in Europe Conference & Exhibition. pp. 894–897. IEEE (2017)
15. Sengupta, D., et al.: Scheduling multi-tenant cloud workloads on accelerator-based systems. In: Supercomputing Conference. IEEE (2014)
16. Vaishnav, A., Pham, K.D., Koch, D.: A survey on FPGA virtualization. In: International Conference on Field Programmable Logic and Applications (2018)
17. Vaishnav, A., Pham, K.D., Koch, D., Garside, J.: Resource elastic virtualization for FPGAs using OpenCL. In: International Conference on Field Programmable Logic and Applications (2018)
18. Vipin, K., Fahmy, S.A.: Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration. In: International Symposium on Applied Reconfigurable Computing. Springer (2012)
19. Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R.: Large-scale parallel collaborative filtering for the netflix prize. In: International Conference on Algorithmic Applications in Management. Springer (2008)