Journal of Signal Processing Systems
https://doi.org/10.1007/s11265-020-01633-z

Evaluation of Static Mapping for Dynamic Space-Shared Multi-task Processing on FPGAs

Umar Ibrahim Minhas[1] · **Roger Woods**[1] · **Georgios Karakonstantis**[1]

Received: 30 April 2020 / Revised: 30 October 2020 / Accepted: 22 December 2020

Abstract

Whilst FPGAs have been used in cloud ecosystems, it is still extremely challenging to achieve high compute density when mapping heterogeneous multi-tasks on shared resources at runtime. This work addresses this by treating the FPGA resource as a service and employing multi-task processing at the high level, design space exploration and static off-line partitioning in order to allow more efficient mapping of heterogeneous tasks onto the FPGA. In addition, a new, comprehensive runtime functional simulator is used to evaluate the effect of various spatial and temporal constraints on both the existing and new approaches when varying system design parameters. A comprehensive suite of real high performance computing tasks was implemented on a Nallatech 385 FPGA card and show 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 due to external memory access latency and bandwidth limitations. The work has been extended by introducing a novel scheduling scheme to enhance temporal utilization of resources when using the proposed approach. Additional results for large queues of mixed intensity tasks (compute and memory) show that the proposed partitioning and scheduling approach can provide higher than 3× system speedup over previous schemes.

Keywords Multi-task processing · Data centres · Space sharing · Scheduling · FPGAs · Cloud computing

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 (HPC), there has been an increased interest in integrating Field Programmable Gate Array (FPGA) technology [9, 18], but its architecture and programming environment present a different resource sharing model when compared to software programmable accelerators. Furthermore, modern HPC tasks depict heterogeneity when executed on software systems, i.e. they show variability in terms of resource requirements such as compute, memory, control, and execution time [24]) and accommodating multiple such heterogeneous tasks at one instance of time in the FPGA space can be challenging. Optimization of the system's resource utilization in time and space when executing these tasks in an area-shared manner on FPGAs, can lead to suboptimal compute density and system throughput.

In software-based systems, a runtime approach can map a task flexibly to any portion of the underlying hardware, incurring a *microseconds* latency context switching between tasks. In FPGAs, the tasks have to be custom designed and mapped spatially onto the device, incurring a reconfiguration overhead and reducing the efficient utilization of FPGA resources [15]. Partially reconfigurable regions (PRRs) can be created that can be configured independently in time and partially in space, allowing dynamic partial reconfiguration (DPR) at runtime. However, these homogeneous PRR regions result in inefficient resource utilization. Whilst researchers have created heterogeneous PRR approaches and multiple bitstreams for a single task via intelligent off-line and runtime PRR design [7], this still fails to address the utilization issue.

This work is an extension of our original conference paper [19], which showed how we were able to overcome

✉ Umar Ibrahim Minhas
umariminhas@gmail.com

Roger Woods
r.woods@qub.ac.uk

Georgios Karakonstantis
g.karakonstantis@qub.ac.uk

[1] Queen's University Belfast, Belfast, UK

the limitations in compute density imposed by PRR, by creating multiple bitstreams per tasks for the HPC tasks. Using the proposed design space exploration (DSE), we were able to estimate the average FPGA resources utilization and thus gauge the effectiveness of various PRR optimizations. We achieved a higher compute density by static partitioning and mapping (SPM) heterogeneous bitstreams, thus providing complete spatial independence for the heterogeneous tasks running 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. Both approaches were compared using a new, comprehensive functional simulator which allowed evaluation of estimated average system *speedup* for runtime processing of large task queues. Moreover, implementation of a number of selected cases on FPGA allowed us to analyze the constraints of both approaches for compute or memory intensive tasks. We analysed the performance in terms of System Throughput (STP), a metric defined specifically for multi-task workload processing.

This research presented here enhances the work through better performance estimation, analysis of targeting dynamic environments, comparison against native FPGA execution and a novel scheduling policy. It makes a number of contributions:

- Extension of the background perspective covering system design perspectives, highlighting the need for such approaches in modern data centres.
- Better performance estimation using memory modelling, analysis of reconfiguration overhead and an extensive comparison against native FPGA execution (NE) of single task configuration per device. The results depict performance of various configuration schemes against dynamics of operating parameters including the mean and deviation of execution time of tasks.
- Creation of a novel scheduling scheme for SPM which uses the single task DSE to enable variation in execution time of tasks to process a workload.
- Use of a clustering algorithm to choose the appropriate design point based on data from the DSE, such that tasks are clustered with similar execution times in order to avoid stalling by the longest running task. The clustered tasks are then co-executed using SPM, thereby enhancing temporal utilization of resources and allowing up to 3× speedup as compared to previous space sharing approaches.

We present an extended background in Section 2 and our updated and revised design and implementation approach is presented in Section 3, with new detail on the runtime simulator, task queue generation, memory performance modelling and input configuration variation. The evaluation environment in Section 4 includes an additional discussion on enabling of DSE in the use cases. Overall, the work now provides a complete flow for implementing static mapping, whilst highlighting the need for evaluation of system throughput whilst deciding if, and how, to partition the FPGA based on the dynamics of operating environment. An extended result section (Section 5) compares both partitioning schemes to NE and includes an analysis of reconfiguration overhead and scheduling policy while incorporating memory modelling. Conclusions are given in Section 6.

2 Background and Motivation

Cloud services are being used by a range of users with diverse computing requirements which vary with task types and workload sizes [26]. In FPGA, the compute versus memory intensity of the tasks, suggests the need for FPGA sharing by heterogeneous tasks in order to achieve maximum system utilization. However, FPGA design tools typically still consider the fabric as a single sea of gates and do not provide any optimum space sharing mechanism. This provides challenges for space shared multi-task processing, restricting commercial cloud services to support only native FPGA execution even with increasing device sizes.

For space shared execution of multiple tasks, the traditional approach is to partition the FPGA into rectangular fixed size PRRs which are configured typically with a new bitstream via DPR, independently of the processing going elsewhere [27]. This provides independence in time for each PRR, such that a task A running in a PRR can be instantly replaced by task B, when finished. This offers low-latency reconfiguration of modules for different stages of a single task or functionally similar tasks as the PRR's design can be custom optimized statically as per the task requirements [4, 5]. However, for large HPC tasks with complex and varying circuit design and I/0, PRR may result in suboptimal utilization of resources.

PRR design is challenging as the spatial distribution of various types of resources on modern tiled FPGA is unsymmetrical, particularly along the horizontal axis (Fig. 1). Furthermore, the FPGA is divided into multiple clock regions across both the vertical and horizontal axes, where the region boundary crossing requires custom logic and cannot be supported by modern runtime bitstream relocation schemes [22]. This limits relocation to homogeneous regions along the y-axis with a step size equal to height of clock region (Fig. 1), in line with work on PRR systems for independent tasks [27].

These mapping constraints require PRRs to be homogeneous along the y-axis, but modern HPC tasks are inherently heterogeneous, i.e. require different number and type of resources (i.e. memory, compute, logic) [6, 24]; mapping

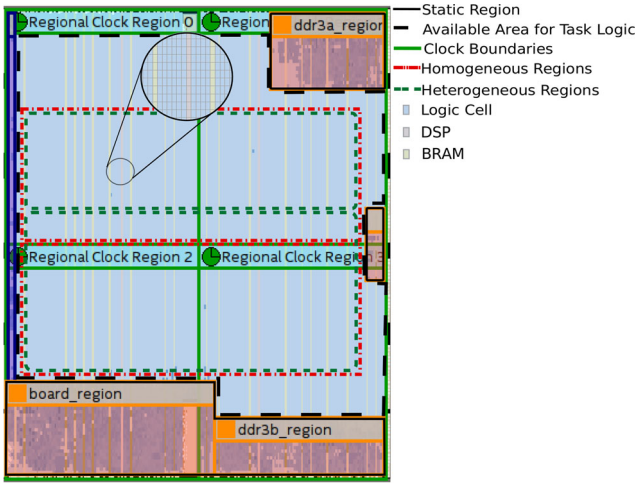

Figure 1 FPGA Partitioning for PRR.

these tasks to homogeneous PRRs may lead to inefficient 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 [27]. 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 3.2.2) after considering static resources and clock regions. 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% [28] which is similar to our own implementation of HPC tasks (Section 5). This is worsened in case of fixed PRRs due to diverse spatial placement of different types of resources.

Previous work has looked to target various optimisations for PRR-based designs. Researchers have explored maximising resource utilization by optimum ordering of tasks [17], task graph based scheduling as per the underlying architecture [23], heuristics to reduce fragmentation within PRRs [12] and runtime support for elastic resource allocation [27]. However, the inherent underutilization of resources when using PRR due to spatial mapping constraints on FPGA remains the same.

Whilst mapping optimizations using PRRs is well researched, this work analyzes for the first time the effect of the constraints of PRRs and inefficient utilization of resources on compute density when mapping heterogeneous tasks. Firstly, we create a large design space using a range of real tasks while exposing the area-throughput trade-off, using the biggest selection of the most relevant HPC tasks to date [8, 25]. This allows us to quantify the heterogeneity in resource utilization by modern workloads when mapping to FPGA and to highlight 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 means 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 of FPGAs in dynamic environments. The approach aims to provide complete spatial independence for highly optimized mapping on account of partial time independence - as all tasks need to be reconfigured at the same time. We quantify this and comment on design parameters that affect system performance while comparing both approaches.

Finally, we propose a novel scheduling approach for executing tasks using SPM to increase the temporal utilization of resources; it uses the DSE to vary resource allocation per task as per the workload. The evaluation of both partitioning schemes and scheduling policy is enabled by a purpose designed, flexible runtime simulator that allows corresponding variation of temporal and spatial system design parameters and constraints.

3 Implementation and Evaluation Methodology

The design and evaluation environment is summarized in Fig. 2. The first step involves defining input tasks using the OpenCL computing model and high-level synthesis parameters to enable the DSE. For any new task to be added to the design environment for optimised space-shared execution, it needs to be manually programmed to expose its area-throughput trade-off via varying implementations.

The designed tools then allow automated exploration of the generated design space in different ways. Firstly, they implement statistical analysis to report the resource utilization efficiency using PRR. Secondly, a runtime functional simulator purposefully designed for comparative evaluation of partitioning schemes is used to explore associated spatial mapping approaches. It takes into consideration various constraints and configurations that affect system performance for multiple partitioning schemes and task queues.

Thirdly, selected tasks are provided as input to the functional simulator to generate configuration for their optimum integrated multi-task designs. These are implemented and profiled on the hardware against varying workload sizes and resource allocation per task to gain insights into system throughput. Finally, the DSE forms the basis of the proposed scheduling policy that targets an improvement in the temporal utilization of resources when using SPM. The implemented scheduling policy is evaluated using the

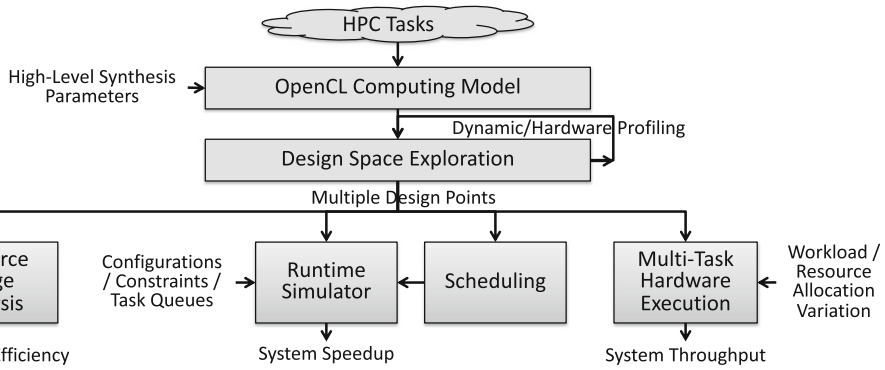

Figure 2 Summary of implementation and evaluation methodology.

function simulator. Various modules are described in more detail below.

3.1 Single Task Design Space Exploration

A key goal is to generate multiple hardware bitstreams of the same task to ensure a *speedup*. This is achieved by undertaking an area-throughput trade-off that gives precise quantification of the variation in compute density against resource utilization. This allows an exploration of the throughput for various multi-task configurations against resource utilization, including mapping efficiency against the total resources available, as discussed in Section 5.

This is a manual process and a necessary step to incorporate new tasks for space-shared execution. Although similar to scaling-out processing on software-based systems such as using threads for parallel processing, the scaling on FPGAs is different due to spatial mapping of code. We describe the adopted process that can be used on incoming new tasks.

We make use of the OpenCL framework for heterogeneous parallel programming as it provides abstraction of parallelism and a high-level DSE model for tuning the underlying hardware mapping. In addition, general high-level synthesis parameters can be used to scale the task over multiple parallel compute units *(CUs)*; multiple pipelines can be defined via a Single Instruction Multiple Data *(SIMD)* parameter; key compute intensive loops can be unrolled via UNROLL *(U)*. For some tasks, we vary task-specific parameters e.g. block size, number of rows, as these define the level of parallel processing. These allow us to scale the number of custom parallel data paths for each task.

For cases where only a part of source code is unrolled, the DSE uses dynamic profiling to identify the compute intensive segments of the kernels and help allocate resources to blocks of code accordingly. As we lack cycle accurate visualisation of execution of OpenCL generated bitstream against the source code and need to incorporate real-time memory performance, an always active counter (written in VHDL) is incorporated to identify bottlenecks and passed to OpenCL kernel as a software library via the Intel OpenCL Library feature. This process is supplemented by manual exploration to identify the parameters that provide the highest throughput variation per unit area.

3.2 Runtime Functional Simulator

An exhaustive runtime functional simulator has been developed. It is the key tool that allows automated evaluation and comparison of various partitioning schemes. It also generates and verify optimum configuration of tasks for hardware implementation of multi-task designs. Furthermore, it incorporates the designed scheduling policy for evaluation against varying design parameters. The simulator is summarised in Fig. 3. All components have access to the single task DSE and the resulting parameters including the utilization of on-chip (logic, DSP, BRAM) and off-chip (bandwidth) resources and corresponding throughput for various generated bitstreams.

Using these parameters, a random task queue is generated. Although the bitstream parameters are used from real benchmarks, the associated execution time can also be generated synthetically against an input mean and deviation to gauge the effect on system performance. For the generated queue, the user can decide if the simulator may use the proposed scheduling policy or map tasks in the order of input.

Once the order of task has been provided, a resource manager performs multi-task DSE to optimise spatial mapping for both PRR and SPM. The resulting designs are provided as input to the runtime placement checker module that ensures that the generated designs can be realistically mapped to the underlying hardware. As input, it takes user configurations indicating the choice of partitioning scheme, PRR optimizations and the reconfiguration overhead of the time that the considered task occupies the device. It also uses spatial mapping constraints, such as the available resource and reconfiguration overhead associated with each design.

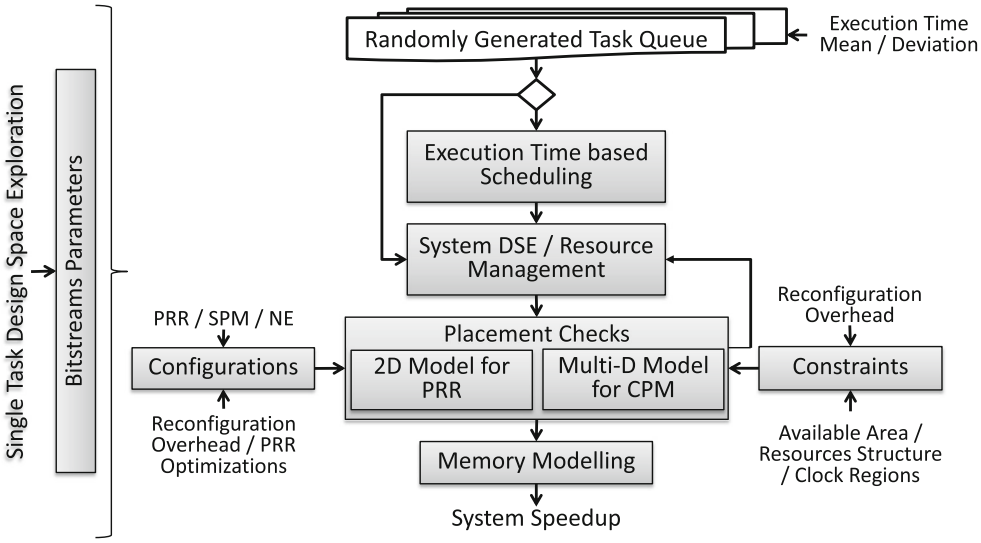

Figure 3 Runtime functional simulator.

Finally, to estimate multi-task performance of generated designs using single task designs profiled on hardware, the simulator uses a memory performance modelling technique to provide the system *speedup* for various configurations, measured as execution time against the set baseline.

3.2.1 Task Queue Generation

The task queue is randomly generated in order to select the task and associated bitstream, by exploring a synthetic execution time for a uniform distribution for which mean and deviation is provided as input. This is needed to explore the impact of stalling by the longest running task while using SPM. Similarly, the mean of the execution time of tasks is required to study the impact of reconfiguration overhead, against the dynamics of task queue.

After the creation of an initial queue, a mapping check verifies that the associated bitstreams can be fitted in PRRs, otherwise they are replaced with smaller bitstreams and their execution time is updated. This update uses the actual difference in throughput, as profiled on hardware, to maintain fairness. In a similar way, bitstreams are replaced with the largest available for each for evaluation of NE.

3.2.2 System DSE / Resource Management

For resource management with SPM, once a multi-task design has been selected, the tool checks if any of the task designs can be updated with a bigger design offering higher throughput. For PRR, the tool implements various optimizations on top of previously explored homogeneous PRRs, allowing us to compare the PRR with SPM. These generate homogeneous regions as well as a single bitstream for each task corresponding to that region. The *Elastic resource allocation* optimization examines adjacent PRR regions and if available, attempts to replace a larger bitstreams of the same task in this combined region, thus achieving a *speedup* [27].

Another approach partitions the FPGA into heterogeneous PRRs with different number of resources [7] [9], thus supporting custom design tasks with different heterogeneous resource allocations. However, the device size is not big enough here 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). This allows us to use smaller (*Contract*) bitstreams for tasks when none of the original bitstream can be fitted into a region [7].

Finally, the simulator allows variation in step size for bitstream relocation beyond the realistic clock region size. This can be varied against the continuous y-axis, i.e. the hypothetical performance gains that can be made if the step size is reduced to a single row by future technology support. At present, this is achieved by generating multiple bitstreams, equal to the number of $rows$ in each clock region, by varying starting y-coordinates for each unique bitstream.

3.2.3 Placement Checks

This module checks if the generated multi-task design can be realistically mapped onto FPGA at any single time. For the PRR, the 2D area model treats mapping as a rectangle fitting problem and tries to find a region homogeneous in both size and spatial distribution of resources to which to map each incoming task [17]. For SPM, a multi-dimensional model accommodating a dimension for each of the heterogeneous on-chip resources, is implemented. The system mapping optimizations try to accommodate as many tasks as possible while meeting the total resources limit.

For PRR, each task's configuration is treated independently while for SPM, a configuration waits for the longest running task before selecting a new multi-task configuration.

3.2.4 Memory Modelling

Even when the allocated resources in the multi-task environment are the same as the single task, the task may not provide the same throughput due to memory contention. To model this, the tool incorporates ridge regression [14] as it provides a lower error when compared to other low complexity linear regression models, e.g. ordinary least squares. As independent variables, it uses the bandwidth usage for single processing mode and a binary value for data access pattern (regular, irregular) of its task and all the tasks with which it is shares the space. Degradation in memory performance for the considered task due to contention is considered as a dependent variable. We generate a range of multi-task bitstreams and measure the actual performance to train the model.

Initial results on the accuracy for area and memory model of both PRR and SPM were presented in [19] but we have now incorporated both of these in the simulator to evaluate system throughput. In particular, the memory modelling affects SPM more due to the greater bandwidth requirement needed in higher compute density designs.

3.2.5 Configurations

The choice of configurations involve use of various partitioning schemes as well as various PRR optimizations. Similarly the reconfiguration overhead can also be switched on or off for varying analysis.

The reconfiguration overhead allows the study of its effect on the total execution time, particularly with variation in mean execution time of tasks due to the dynamics of different environments such as edge vs cloud, etc. The significance of reconfiguration overhead reduces with larger workloads requiring higher execution times. Furthermore, the reconfiguration overhead is directly related to the area being mapped. Thus as the throughput increases with more resources when going from PRR to SPM and NE, the gains may be offset by the higher reconfiguration overhead.

3.2.6 Constraints

In PRR, the homogeneous/heterogeneous regions are fixed and the coordinates are provided by the user along with mapping restrictions due to clock regions. In SPM, the total number of available heterogeneous resources and a realistic percentage of the maximum utilization, as found during DSE, is provided as input. To study the effect of various PRR constraints, the available area for the task mapping can be varied.

3.3 Hardware Implementation

To evaluate compute density, multi-task bitstreams for a select number of cases have been generated and profiled on hardware against varying workload sizes. OpenCL is used to generate intermediate design files followed by placement scripts to constrain modules to corresponding areas on the FPGA. For SPM, this includes all of area available for task logic in Fig. 1. We do not implement any further partitioning optimizations on top of those inherently provided by the used Intel tools (for Nallatech board). For PRR, the placement scripts constrain each task module to one of the homogeneous PRR as described in Fig. 1. Once the constraint file has been updated, the Quartus Prime synthesis and bitstream generation tool is used to generate the final bitstreams and then to map the largest possible bitstream configurations for both of PRR and SPM within their respective area constraints.

3.4 Scheduling

SPM can provide better resource utilisation and thus, higher throughput but the multi-task designs are statically implemented to create a single integrated bitstream. The multi-task bitstream is configured on the FPGA as a single unit. However, different tasks in that bitstream can have varying workloads and hence, different execution times. This means that the integrated bitstream can only be reconfigured with a different one once all tasks sharing the space have finished execution. In a dynamic environment with a range of tasks and associated variable workloads, the stalling of execution every time by the slowest task in the multi-task bitstream can result in significant underutilization of resources.

To counter that, the work proposes a novel pre-emptive scheduling approach which *clusters* queued tasks based on their pre-empted execution times to provide a feasible schedule. The idea is to ensure that all tasks being executed in a space shared fashion have similar execution times so that they start and finish processing at a similar time. This is ideally suited to SPM and has been validated using the functional simulator to ensure the reduction of stalling by the longest running task for integrated bitstreams.

The approach starts with the design space created by single task exploration and then varies the resource allocation per task. To maintain similar execution times for tasks in a cluster, each task's execution time is determined by the resources allocated to it. Furthermore, to enable pre-emptive scheduling, variable sized data blocks of tasks are profiled off-line for all bitstreams where the block refers

to input sizes batch processed in an OpenCL like model. It is presumed that the incoming workload requests can be represented as a summation of one of these block sizes. Thus by knowing the estimated task's execution time for each design, this allows us to select the appropriate one and then co-execute it with other tasks with similar execution times.

Using k-means clustering [13], we explore a single feature-based approach based on execution time. Data is clustered by first computing centroids based on the defined number of clusters, and then placing items in the relevant cluster based on their Euclidean distance from the centroid. We execute both these steps separately as explained next.

In the first step, execution times for all tasks in an incoming queue are clustered using all of their bitstreams. Only one bitstream is eventually used for execution, however, using all of them increases the design space. This results in a centroid set, $\{c_1, c_2, c_3...c_n\}$. The tasks are then assigned clusters. For each task, one of the bitstreams is selected, such that the relative distance of corresponding execution time from the closest centroid, c_i, defined as a percentage of centroid value, is less than the relative distance for any other bitstream from their closest centroid. A second level of k-means clustering is then run on these newly generated clusters. Finally, an outlier filtering function is executed where tasks still having a distance greater than threshold deviation from respective centroids, are put in a separate cluster and executed in a native FPGA configuration.

To enable runtime selection of appropriate clusters, the scheduler should have access to pre-generated multiple SPM bitstreams offering varying combinations of tasks as well as resources allocation per task. This can be optimised by using workload characterisation of servers to predict the dynamic requirements. Furthermore, new integrated bitstreams can be generated in parallel as the requirements arise while the scheduler makes use of the available options.

4 Evaluation Environment

The evaluation considers 10 HPC tasks belonging to various computing dwarfs and application domains [3]. The tasks have been chosen from various cloud and HPC benchmarks targeting accelerated computing in data centre environments [8, 16] as they feature a good range of challenging, heterogeneous computing characteristics. Here, the high-level parameters used for each task for DSE are briefly explained, without going into algorithmic details as further explanation is given in relevant quoted references. Various parameters scaled for each task are summarized in Table 1.

a) Alternative Least Squares based Collaborative Filtering (ALS) has been applied in a recommender system for commercial domains such as Netflix [29]. The DSE on ALS is performed using the number of CUs for the kernel as well as loop unrolling for the *error calculation in recommendation estimation* after each iteration.

b) Page Rank (PR) is a graph analysis algorithm used for link analysis of web pages, social networks, etc [21]. Bitstream scaling is performed by exploring variation in the number of CUs and loop unrolling in order to calculate a new $rank$ for each $page$.

c) 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 [20]. The DSE involves allocating more CUs and the unrolling of loop transversing the $binomial\ tree$.

d) Lower Upper Decomposition (LUD) is an important dense linear algebra used for solving systems of linear equations with reduced complexity [8]. Area-throughput trade-off is generated by varying number of CUs and loop unrolling of the compute intensive loops in $decomposition$.

e) Breadth First Search (BFS) is one of the most challenging and important graph traversal algorithm which forms the basis of many graph-processing workloads [8]. The hardware for BFS is scaled by unrolling the loop that transverses $edges$ of a single $node$.

f) 3 Dimensional Finite Difference Time Domain's (FDTD) implementation [1] uses a window to slide through the space with all points in the window being processed in parallel. The number of $points$ in a window are used to scale underlying hardware.

g) Sparse Matrix Vector Multiplication's (SpMV) is an important sparse linear algebra algorithm used in scientific applications and graph analytics, etc [11]. The underlying hardware is scaled by unrolling the loop that calculates the sum for each row.

h) Matrix-Matrix Multiply (MM) is used in various compute intensive algorithms and benchmarks [11]. This uses the *SIMD pragma* as well as unrolling of loop calculating $dot\ product$ to scale the hardware utilization.

i) Video Downscaling (VD) is used by a range of media streaming services for real-time bandwidth reductions [1]. The variation in number of $rows$ of $pixels$, being processed in parallel using channels, allows scaling of the resource utilization and throughput.

j) Needleman-Wunsch (NW) is a bioinformatics optimisation algorithm used for protein sequence alignment [8]. NW's implementation scales the underlying hardware by varying the size of $strings$ that it uses to divide the bigger problem and process in parallel on FPGA.

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

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

4.1 Metrics

Assessing the system performance of a multi-task workload running in parallel on a single processing unit is challenging, as the absolute measure of individual tasks' throughput does not provide an indication of system performance. This is because the contribution to absolute processing time and average speedup may be dominated by tasks with larger workload sizes, so uniform metrics e.g. FLOPS will not provide a meaningful measure for all of the tasks being evaluated.

We use two different metrics for simulated and hardware results in order to allow a realistic and comprehensive assessment to be made. Firstly, the simulation of large task queues provides the potential to estimate an average *speedup*, measured as the variation in execution time of the complete queue. To evaluate the compute density provided by various approaches in a multi-task environment, the STP metric [10] is used and is 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 normalised progress defined by the number of clock cycles taken in single task mode, C_i^{SP}, when the task can avail of all of the FPGA's resources as compared to multi-task mode, C_i^{MP}, when it shares the space with other tasks. Here, n defines the number of tasks sharing the FPGA.

4.2 System Hardware

The DSE has been coded using Intel's OpenCL SDK for FPGAs v 16.1 and implemented on a Nallatech 385 which uses an Intel Stratix V GX A7 FPGA and 8GB DDR3 memory. The A7 chip has 234,720 ALMs, 256 DSPs and 2,560 M20K BRAM blocks. Runtime simulations were performed using Python v3.3.7 running on a single core of $Intel\ Xeon\ CPU\ \ E5-1620\,v3@3.50GHz$ (host).

5 Results And Analysis

The baseline throughput is defined by the serial pipelined implementation and corresponds to the lowest area bitstream whereas the maximum throughput is defined by the largest possible bitstream. We have generated $4-9$ bitstreams per task providing $2-204\times$ maximum $speedup$ compared to slowest bitstream as shown in Table 1. The parameters used for parallelism are also highlighted e.g. number of compute units, unrolling of main computing loop, use of 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 will demonstrate.

5.1 Analysis of Heterogeneous Tasks

Using the DSE, we analyze the heterogeneity in on-chip 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

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%
DSPs	32.33%	0.0% / 97.0%	26.30%	0.0% / 97.0%
Block RAM	60.56%	15.49% / 95.82%	45.05%	10.07% / 91.91%

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.

5.2 Runtime Functional Simulation

In this section, we use our simulator to analyse various mapping strategies.

5.2.1 SPM vs PRR

Firstly, we examine the *speedup* achieved by various improvements on the PRR mapping, as explained in Section 3.2.2. 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.

The simulator's resource manager performs Elastic and Contract optimizations, as explained in Section 3.2.2. We use the actual measured relative throughput of various bitstreams of tasks to calculate the new execution time of tasks. For Contract, we determined that if the difference in *speedup* for a smaller bitstream replacing the bigger is too large, the total execution time increases rather than decreases. Thus, we limited the allowed *speedup* degradation for smaller bitstreams to 5×.

The graphs in the Fig. 4 show the *speedup* achieved by the optimizations for various configurations. Generally, Elastic is more useful with gains up to 1.33× whereas the best gain for Contract is 1.05×. Optimizations benefit more with heterogeneous mapping in tackling segmentation; hence, the gain is negligible for our case with only two heterogeneous regions while no gain is achieved for homogeneous regions.

Next we investigate gains made by SPM in comparison to PRR. For SPM, we either use the same region as used for PRR (Homogeneous Regions in Fig. 1) and call it P-SPM or use whole of the available area for task logic (Fig. 1) and call it W-SPM. 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 indicate, a key finding is that SPM gives on average 4.2× higher throughput, measured in terms of total execution time for a set size of task queue. A 2.2× *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 1.8× can be achieved.

5.2.2 Execution Time Variation

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, having the same execution

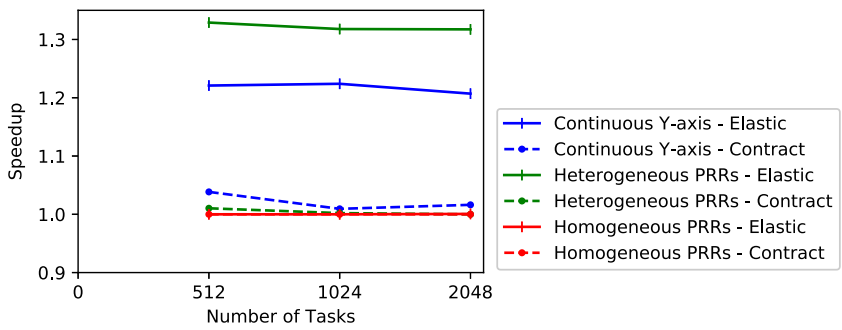

Figure 4 *Speedup* achieved by optimization of PRR mapping on various bitstreams.

Figure 5 *Speedup* achieved by SPM versus PRR mapping.

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. Also we now show results against native FPGA implementation.

The results shown in Fig. 6 for speedup depict a surprising trend. Even with increasing the range of execution time, defined as a multiple (×) and represented here by *Deviation (Dev)*, of up to 32× (beyond this range a reconfiguration overhead to replace the bitstream would become negligible), the *speedup* by SPM decreases but remains higher than 2.5× that of PRR. 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 are overcome by the higher average compute density and gains made when the longest running task are not the smallest. The speedup for PRR is maintained at 0.38×.

The trend for SPM against NE follows a similar trend. However, although there is an initial gain of 1.5×, the speedup falls below 1 after an execution time *Dev* of 8×. This suggests that even though SPM provides denser spatial mapping than NE, the suboptimal temporal utilization can degrade performance. There are other benefits to sharing the FPGA device particularly for smaller tasks, however, the work argues that throughput analysis should be performed against the dynamics of the considered task queue to select the optimum mapping scheme. Next, we analyze the dynamics of task queue in the context of reconfiguration overhead.

5.2.3 Reconfiguration Overhead

So far, we considered reconfiguration overhead to be negligible as compared to task execution time and hence, did not include it in the analysis. It is included in the next set of experiments which show the total execution time for processing a queue of 1024 tasks. The reconfiguration time for PRR is calculated as percentage of the total reconfiguration time for the entire FPGA resources. SPM, similar to NE, is reconfigured as a single FPGA bitstream.

For 1024 tasks, NE is reconfigured 1024 times with a total reconfiguration time of 1996.8 seconds. We use two homogeneous PRRs for this set of experiments resulting in total reconfiguration time of 559.1 seconds. Finally, SPM

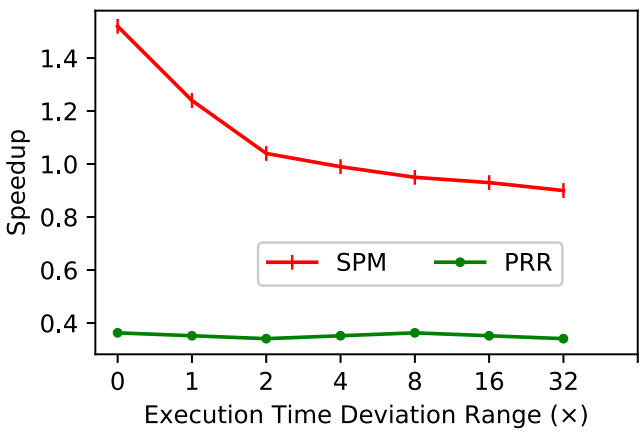

Figure 6 *Speedup* Variation of SPM and PRR with variation in execution times against NE (Tasks = 1024).

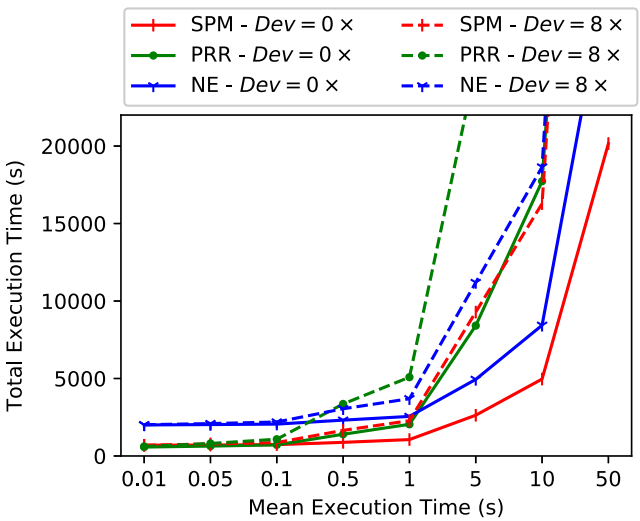

Figure 7 Total execution time including the reconfiguration overhead for varying mean of tasks individual execution time.

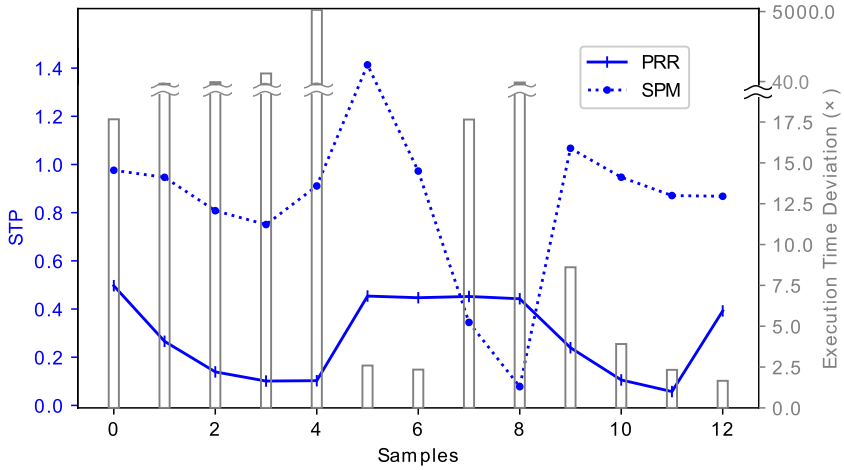

Figure 8 *STP* Variation with data sizes for 2 compute intensive tasks - MM and LUD.

can take variable number of reconfigurations to process a fixed number of tasks based on the individual sizes of tasks and runtime resource management. In our experiments, the number of reconfigurations ranged from 363 to 379 with a reconfiguration overhead of 672 to 702 seconds.

The total execution time for various mapping schemes against a varying mean execution time of tasks is shown in Fig. 7. It includes two different ranges for each evaluated mean. Starting from the offset of reconfiguration overhead, the total execution time generally increases linearly with increase in mean execution time of tasks. However, the rate at which the reconfiguration overhead becomes less significant while the task processing time becomes more significant towards total execution time, varies with the mapping scheme. The lower reconfiguration overhead plays more significant role towards better performance for smaller tasks with lower mean execution time while higher throughput is more significant for larger tasks.

In the first test scenario, the range of execution time approaches zero or in other words, all tasks have similar execution times. In this case, even with the lowest throughput associated with PRR, it provides the best overall system performance by up to 1.2× owing to the lowest reconfiguration overhead. However, the lowest throughput and hence the highest task processing time for PRR becomes the more significant factor towards total execution time quickly with the increasing task size and the PRR becomes the worst performing for tasks taking more than 1 second per task.

The second set of experiments shows the benefits of space sharing when using an increased 8× range of the execution times (*Dev*). Without considering reconfiguration overhead, SPM performed worse than NE as shown in Fig. 6. However, space sharing and lower number of resources per task results in lower reconfiguration overhead for SPM compared to NE. This results in SPM providing better overall performance by an average of 1.8×. The overhead only offsets the performance loss due to lower throughput though, as SPM may perform worse than NE with even higher *Dev*. Fig. 7 provides

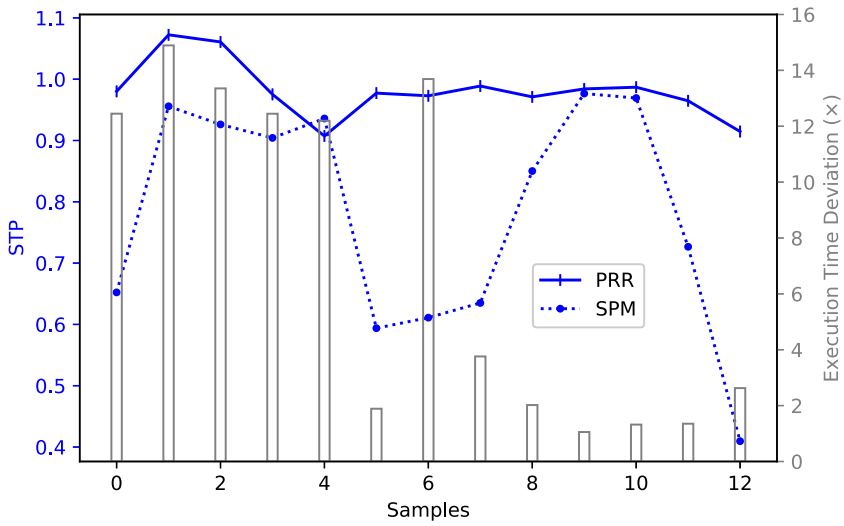

Figure 9 *STP* Variation with data sizes for 2 memory intensive tasks - ALS and PR.

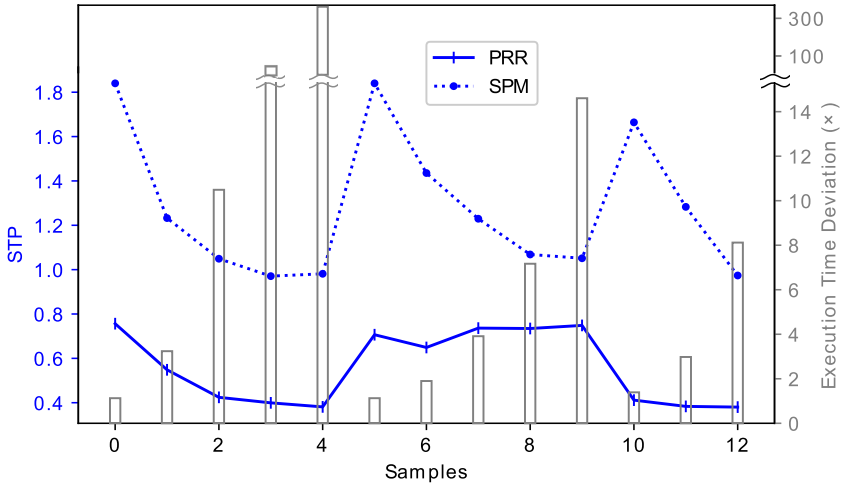

Figure 10 STP Variation with data sizes for one compute and one memory intensive task - LUD and PR.

a generic picture of how the mean and the range of execution time of tasks may affect selection of the optimal mapping scheme and may be considered when designing a system.

5.3 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 [2, 24]. 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. Apart from helping with a higher resource utilization on-chip, these decisions minimize 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 4.1.

In terms of resource utilization, SPM resulted on average 60%, 129% and 59% higher logic, DSP and BRAM utilization compared to PRR, respectively. Furthermore, Figs. 8, 9 and 10 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 Dev 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 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, as 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

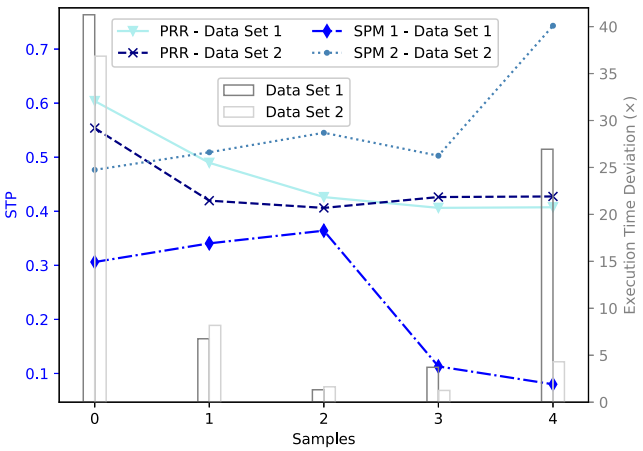

Figure 11 STP Variation for PRRs and SPM for two configurations using four tasks - SPMV, NW, LUD, PR.

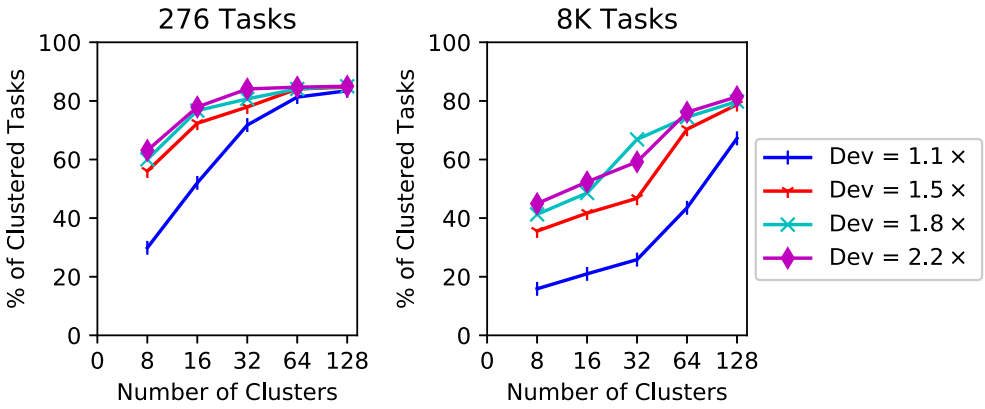

Figure 12 Percentage of tasks clustered for different number of steps.

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 steps of 2×). The resulting STP presented in Fig. 11 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.

5.3.1 Accuracy of Performance Estimation Models

Here, we provide an analysis of the accuracy of the multi-task performance estimation model. To evaluate the model, we generated a set of binaries for various combinations and scale of tasks for both PRR and SPM. The memory model was trained on binaries different from the ones used for testing. For PRR, the model generated, on average, a root mean square error of 3.15% with a worst case error of 9.8%, while for SPM an average error of 5.5% and a worst case of 11.6% was observed. The error is calculated for the estimated relative speedup in multi-task processing using the NE in single task processing and compared against the actual performance in multi-task execution for the tested cases.

5.4 Scheduling

So far, the studies have used real bitstream parameters but with synthetically generated execution times. For the next set of experiments that evaluate scheduling policy, real execution time for various sizes of workload were used. The first step of the scheduling policy is to create clusters. At this stage, the tunable parameters are the number of used clusters and allowed deviation of execution time in a single cluster. Both factors, in turn, affect the number of tasks per cluster where the higher value gives more mapping options to the runtime scheduler, leading to a higher compute density in space. The higher variation in execution times in a cluster has the impact of resulting in more stalling by the longest task in a package, thereby giving higher compute density in time. However, deviation and number of tasks per cluster are directly proportional as larger deviation allows more tasks in a cluster or in other words, requires less clusters.

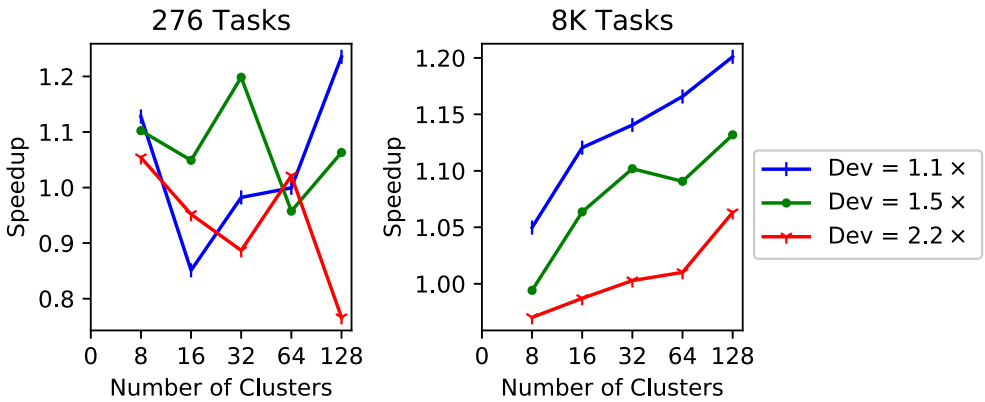

Figure 13 Speedup using clustered tasks against native FPGA execution.

Figure 12 gives results on clustering on real data for two varying task queue sizes where each task represents a unique workload size. The figure highlights the same trends as above where the higher deviation and number of clusters result in higher number of tasks being assigned to clusters. Furthermore, the number of tasks per cluster goes up with increasing task queue size. For example, at 1.2× *Dev* and 32 clusters, there are on average 6 tasks per cluster for a task queues of 276. However, the same parameters result in 64 tasks per cluster for 8K task queues. This suggests that the approach can gain more on larger scale systems with bigger task queues.

By increasing *Dev*, the number of clustered tasks increases slightly, but the effect on performance drops significantly due to longer stalling by larger tasks and hence it is not considered. Our two step clustering and filtering method increased the number of clustered tasks by up to 2.3× compared to single step clustering. The clustering and filtering took 0.47s - 2.5s and 5.68s - 70s for 8 - 64 clusters for the 276 and 8K task queues respectively, using a non-optimized, non-parallel implementation running on the host.

The runtime scheduler maps the task queue while maximising resource utilisation. During the clustering process, some of the clusters end up with a single task cluster. At runtime, the bitstreams for these, along with for outliers tasks, are switched to the largest bitstream for the corresponding task to run as single task per device. On a multi-FPGA cluster, these can also run on a separate FPGA in a non-clustered configuration.

The speedup gained against a NE configuration is shown in Fig. 13. The smaller task queue does not provide a clear picture as the performance varies largely by fine variation of parameters and the change in order of tasks in the queue. The larger task queue shows that although larger deviation allows more tasks to be clustered, the effect on overall performance is detrimental. Similarly using more clusters also improves the net performance. Overall, the average gain of 1.13× for *Dev* of 1.1× is lower than the results shown earlier, because the gain is only possible in the clustered tasks. The number of clustered tasks for a set *Dev* may be increased by using a larger device that offers more design points. The speedup for PRR stands constant at 0.38× and thus SPM can provide 3× higher speedup for space-shared multi-task execution.

6 Conclusion

This work analyzes the constraints of mapping bitstreams 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 DSE using a range of HPC tasks, a comprehensive simulator and evaluation on hardware. A novel scheduling policy utilising execution times of tasks to cluster for co-execution is proposed to increase temporal compute density. Static partitioning and scheduling policy provides up to 3× higher system throughput and facilitates a completely software based implementation of a multi-task space shared computing environment without requiring low level support for PRR. The work also highlights that throughput must be considered and evaluated against the dynamics of task queue while deciding if to partition and how to partition.

References

1. Developer Zone. Intel FPGA SDK for OpenCL. https://www.intel.co.uk/content/www/uk/en/programmable/products/design-software/embedded-software-developers/opencl/support.html (2018).
2. Abdul-Rahman, O.A., & Aida, K. (2014). Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon. In *International conference on cloud computing technology and science*: IEEE.
3. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W., et al. (2006). The landscape of parallel computing research: a view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.
4. Banerjee, S., Bozorgzadeh, E., Dutt, N.D. (2006). Integrating physical constraints in HW-SW partitioning for architectures with partial dynamic reconfiguration. *IEEE Transactions on VLSI Systems*, *14*(11), 1189–1202.
5. Cattaneo, R., Bellini, R., Durelli, G., Pilato, C., Santambrogio, M.D., Sciuto, D. (2014). Para-sched: a reconfiguration-aware scheduler for reconfigurable architectures. In *IEEE International parallel & distributed processing symposium workshops* (pp. 243–250).
6. Chang, X., Xia, R., Muppala, J.K., Trivedi, K.S., Liu, J. (2016). Effective modeling approach for IAAS data center performance analysis under heterogeneous workload. *IEEE Transactions on Cloud Computing*, *6*(4), 991–1003.
7. Charitopoulos, G., Koidis, I., Papadimitriou, K., Pnevmatikatos, D. (2017). Run-time management of systems with partially reconfigurable FPGAs. *Integration, the VLSI Journal, 57*.

8. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.H., Skadron, K. (2009). Rodinia: a benchmark suite for heterogeneous computing. In *IEEE International symposium on workload characterization* (pp. 44–54).
9. Chen, F., Shan, Y., Zhang, Y., Wang, Y., Franke, H., Chang, X., Wang, K. (2014). Enabling FPGAs in the cloud. In *Proceedings of ACM conference on computing frontiers*.
10. Eyerman, S., & Eeckhout, L. (2008). System-level performance metrics for multiprogram workloads. *IEEE Micro, 28*(3).
11. Gautier, Q., Althoff, A., Meng, P., Kastner, R. (2016). Spector: an openCL FPGA benchmark suite. In *IEEE International conference on field-programmable technology* (pp. 141–148).
12. Gu, Z., Liu, W., Xu, J., Cui, J., He, X., Deng, Q. (2009). Efficient algorithms for 2D area management and online task placement on runtime reconfigurable FPGAs. *Microprocessors and Microsystems*, *33*(5-6), 374–387.
13. Hartigan, J.A. (1975). *Clustering algorithms*. New York: Wiley.
14. Hoerl, A.E., & Kennard, R.W. (1970). Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.
15. Huang, M., Wu, D., Yu, C.H., Fang, Z., Interlandi, M., Condie, T., Cong, J. (2016). Programming and runtime support to blaze FPGA accelerator deployment at datacenter scale. In *Proceedings of ACM symposium on cloud computing* (pp. 456–469).
16. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B. (2010). The HiBench benchmark suite: characterization of the Mapreduce-based data analysis. In *IEEE International conference on data engineering workshops* (pp. 41–51).
17. Liang, H., Sinha, S., Zhang, W. (2018). Parallelizing hardware tasks on multicontext FPGA with efficient placement and scheduling algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(2), 350–363.
18. Minhas, U., Russell, M., Kaloutsakis, S., Barber, P., Woods, R., Georgakoudis, G., Gillan, C., Nikolopoulos, D., Bilas, A. (2018). Nanostreams: a microserver architecture for real-time analytics on fast data streams. *IEEE Transactions on Multi-Scale Computing Systems*, *4*, 396–409.
19. Minhas, U.I., Woods, R., Karakonstantis, G. (2019). Evaluation of FPGA partitioning schemes for time and space sharing of heterogeneous tasks. In *International symposium on applied reconfigurable computing* (pp. 334–349).
20. Minhas, U.I., Woods, R.F., Karakonstantis, G. (2018). Exploring functional acceleration of opencl on FPGAs and GPUs through platform-independent optimizations. In *International symposium on applied reconfigurable computing* (pp. 551–563).
21. Page, L., Brin, S., Motwani, R., Winograd, T., et al. (1998). The Pagerank citation ranking: bringing order to the web.
22. Pham, K.D., Horta, E., Koch, D. (2017). Bitman: a tool and API for FPGA bitstream manipulations. In *IEEE design, automation & test in europe conference & exhibition* (pp. 894–897).
23. Redaelli, F., Santambrogio, M.D., Memik, S.O. (2009). An ILP formulation for the task graph scheduling problem tailored to bi-dimensional reconfigurable architectures. *International Journal of Reconfigurable Computing*.
24. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A. (2012). Heterogeneity and dynamicity of clouds at scale: google trace analysis. In *Proceedings of ACM symposium on cloud computing*.
25. Sengupta, D., Goswami, A., Schwan, K., Pallavi, K. (2014). Scheduling multi-tenant cloud workloads on accelerator-based systems. In *IEEE Supercomputing conference*.
26. Vaishnav, A., Pham, K.D., Koch, D. (2018). A survey on FPGA virtualization. In *IEEE International conference on field programmable logic and applications* (pp. 131–1317).
27. Vaishnav, A., Pham, K.D., Koch, D., Garside, J. (2018). Resource elastic virtualization for FPGAs using openCL. *International Conference onField Programmable Logic and Applications (FPL)*.
28. Vipin, K., & Fahmy, S.A. (2012). Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration. In *International symposium on applied reconfigurable computing*: Springer.
29. Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R. (2008). Large-scale parallel collaborative filtering for the Netflix prize. In *International conference on algorithmic applications in management* (pp. 337–348): Springer.

Umar Ibrahim Minhas received the B.S. degree from Institute of Space Technology, Pakistan in 2010, the M.Sc. from Imperial College London in 2013 and the Ph.D. from Queen's University Belfast. He was a Research Fellow at Queen's University Belfast at the time of this work. His research interests include energy-efficient processing on heterogeneous SoCs and accelerator-based systems for low-power embedded and high performance computation, respectively.

Roger Woods received the B.Sc. degree (Hons.) in Electrical and Electronic Engineering and the Ph.D. degree from Queen's University Belfast in 1985 and 1990, respectively. He is currently full Professor and Dean at the university with his research interests being heterogeneous computer hardware, data analytics, and wireless communications. He holds four patents and has authored over 230 papers. He is a member of the IEEE Signal Processing and Industrial Electronics Societies and sits on the Advisory Board for the IEEE SPS Technical Committee on the Design and Implementation of Signal Processing Systems. Roger is on the Editorial Board for the Journal of VLSI Signal Processing Systems and the IET Proceedings on Computer and Digital Techniques and serves on the program committees of a number of IEEE conferences. In 2007, he co-founded a spin-off company, Analytics Engines Ltd. which develops data analytics software.

Georgios Karakonstantis is an Associate Professor at the School of Electronics, Electrical Engineering and Computer Science of Queen's University Belfast, United Kingdom and a Senior Member of the IEEE. His research focuses on dependable energy-efficient computing and storage architectures, where he has published more than 85 papers and authored three book chapters. He is the inventor of a US patent, recipient of the IEEE DATE 2020 best paper award and of three HiPEAC paper awards and nominee of the best paper award in IEEE MICRO 2020 and IEEE DATE 2016. He was awarded a Marie-Curie Fellowship by the European Commission on 2012 and won the Altera Innovate Design Contest 2010. He received the MSc and PhD degree in Electrical and Computer Engineering from Purdue University, West-Lafayette, USA. In the past he worked at the Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland, and the Advanced Technology Group, Qualcomm Inc., San Diego, CA, USA. He is the Associate Editor of the IEEE Open Journal on Circuits and Systems and serves as member of the program committee and referee of major IEEE and ACM conferences and journals.