This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

Efficient, Dynamic Multi-task Execution on FPGA-based Computing Systems

U. I. Minhas, R. Woods *Senior Member, IEEE*, D. S. Nikolopoulos *Senior Member, IEEE* and G. Karakonstantis *Senior Member, IEEE*

Abstract—With growing Field Programmable Gate Array (FPGA) device sizes and their integration in environments enabling sharing of computing resources, such as cloud and edge computing, there is a requirement to share the FPGA area between multiple tasks. The resource sharing typically involves partitioning the FPGA space into fix-sized slots. This results in suboptimal resource utilisation and relatively poor performance, particularly as the number of tasks increase. Using OpenCL's exploration capabilities, we employ clever clustering and custom, task-specific partitioning and mapping to create a novel, area sharing methodology where task resource requirements are more effectively managed. Using models with varying resource/throughput profiles, we select the most appropriate distribution based on the runtime, workload needs to enhance temporal compute density. The approach is enabled in the system stack by a corresponding task-based virtualisation model. Using 11 high performance tasks from graph analysis, linear algebra and media streaming, we demonstrate an average $2.8\times$ higher system throughput at $2.3\times$ better energy efficiency over existing approaches.

1 Introduction

Computing services, such as cloud data centres, look to achieve a higher return on cost of computing systems by efficient temporal and spatial sharing of resources amongst multiple tasks. In the recent time, the drive for enhanced performance for computationally demanding tasks have encouraged cloud service providers, such as Amazon, to integrate Field Programmable Gate Arrays (FPGAs) in their data centres [**?**]. Whilst FPGAs offer acceleration in high performance computing (HPC), this is typically restricted to a single application configuration (SAC) [**?**] and effective resource management for area-shared multi-task execution still remains challenging. This has a higher significance, particularly with the migration of services to fog/edge under stricter resource and energy utilisation constraints.

Conventional software-programmable systems typically use discrete processing cores, abstracted as software threads offering microsecond latency context switching between tasks. This results in a highly flexible resource management and tasks scheduling model, to which a range of optimisations can be applied [**?**]. With FPGAs, however, this is complicated as support for mapping source code has to be considered spatially, i.e., how the devices physical resources are shared.

Typically, frameworks are based on partial reconfigurable regions (PRRs) [**?**], [**?**] where the FPGA is partitioned into fix-sized rectangular slots. Spatial mapping constraints on modern FPGAs mean that the PRRs are designed to be largely homogeneous. Modern HPC tasks, however, are inherently heterogeneous where resource needs such as memory, computing, bandwidth, will change and have runtime varying workload sizes and throughput needs [**?**]. Mapping these independent HPC tasks with custom designed hardware and I/O onto PRR systems, typically results in a mismatch giving lower compute density and underutilisation of FPGA resources by up to 70% [**?**].

Custom task-specific partitioning and mapping (CPM), supported by the vendor tools, can be used to generate a single bitstream supporting multiple tasks acceleration functions for area-shared execution. The diverse computing requirements and workload sizes of dynamic task queues make the resource management and tasks scheduling an NP-complete problem, in addition to the long FPGA synthesis times. Thus, any effective approach needs to enable a runtime model that can offload appropriate designs onto FPGA to optimise overall performance, defined as the system throughput (STP) metric [**?**], a more effective measure of performance in multi-task workload processing. A full system stack is also needed for using FPGAs as a resource for optimum execution of dynamic workloads.

In this paper, a task-based implementation, optimisation and execution framework is proposed to tackle these challenges. Using OpenCL's exploration capabilities, a large design space is generated and explored using machine learning to custom generate high compute density and runtime scalable accelerator functions. These are used in an effective manner in runtime, to suit the variable workload requirements, eventually leading to a higher STP for a range of heterogeneous tasks. The main contributions are given as:

- A comprehensive runtime evaluation tool that provides early design analysis of the spatial and temporal constraints of various mapping schemes. It allows selection of the optimal under the operating conditions and objectives, as well as hinting towards areas for optimisation, e.g., area-shared multi-task execution, via functional emulation.
- Machine learning based characterisation of

- *U. I. Minhas, R. Woods and G. Karakonstantis are with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Belfast, United Kingdom*
  E-mail: u.minhas, r.woods, g.karakonstantis@qub.ac.uk
  D. S. Nikolopoulos is with Department of Computer Science, Virginia Tech, Blacksburg, Virginia, United States
  E-mail: dsn@vt.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

area/throughput rate for different tasks and clustering of tasks for custom mapping and co-execution in an area-shared fashion to achieve high spatial compute density designs.
- Multi-task design space exploration (DSE) and use of preemptive scheduling to enable an effective runtime resource allocation based on the size of each co-executed task, in achieving a high temporal density.
- Realisation of a task-based virtualised resource allocation model to support this task-specific area sharing model at the higher system level.
- Use of HPC examples from graph analysis, linear algebra, media streaming and data mining, an average *STP* improvement of $2.8\times$ at $2.3\times$ better energy efficiency is achieved over PRR.

The paper is organised as follows. Section **??** discusses the motivation and background. Section **??** describes our proposed framework, whilst Section **??** gives details of the evaluation environment. Section **??** provides an analysis using the examples and conclusions are given in Section **??**.

2 Background

FPGA-based computing systems: With FPGAs integration in cloud and data centres, work is being carried out to allow seamless and efficient access to the FPGA resources from software environments. Among these, the authors in [**?**] treat the FPGA as an independent resource and develop communication stack and associated hardware on the FPGAs to directly communicate with other CPUs and FPGAs in the cluster. Work in [**?**] uses middleware to provide an application-centric interface for developers and takes care of hardware development and integration for various vendors to enable portability and better productivity. Authors in [**?**] have developed a compilation framework, communication interfaces and runtime libraries to scale resources from sub-FPGA to multi-FPGAs per task requirements. Similarly, work in [**?**] takes a modular approach to system stack design where different components allow portability to integrate with varying versions of design tools and hardware and software layers.

FPGA as reconfigurable resource: Furthermore, with the use of an FPGA as a re-programmable source in dynamic environments, researchers have investigated various reconfiguration and task mapping schemes. The temporal sharing include the reconfiguration of FPGA with a single task (SAC) [**?**], [**?**] and the use of software programmable soft-cores [**?**]. For larger tasks, researchers have looked at providing transparent access to multiple FPGAs with each FPGA processing part of the task [**?**]. In addition, with increasing device sizes, research has explored space sharing, mostly via PRR [**?**], [**?**]. Finally, authors in [**?**] have looked to combine homogeneous PRR block-based design with supported interconnections that scale resources for a task on and across FPGAs transparently.

FPGA space sharing constraints: This work is targeted at tasks that can benefit from sharing of FPGA space. Traditionally this has been achieved via PRRs [**?**], [**?**], [**?**] as it mimics discrete computing resources in multi-core software programmable systems and provides independence in space and time. The independence in time suggests that a task in a single PRR slot can be independently reconfigured with a new task without affecting the processing in other PRR slots, enabling easy scheduling decisions such as task priorities. However, the flexibility in resource scaling per task and achieved resource mapping density suffers.

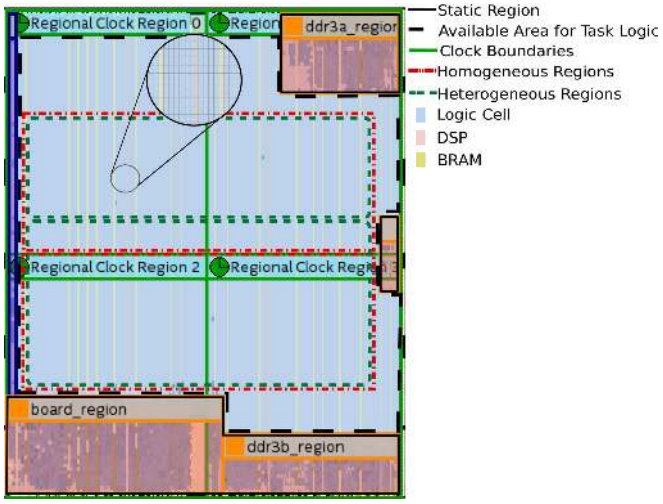

Fig. 1. FPGA Partitioning for PRR

This is because the space domain programming and mapping of FPGA fabric via PRRs is challenging, since the distribution of the heterogeneous resources on modern tiled FPGA is unsymmetrical, particularly along the horizontal axis, as shown in Fig. **??**. Furthermore, the FPGA uses multiple clock regions across both the vertical and horizontal axes and so crossing the region boundary requires custom logic which cannot be supported by modern runtime bitstream relocation schemes [**?**]. This limits relocation to homogeneous regions along the y-axis with a step size equal to height of clock region (Fig. **??**), in line with work on PRR systems for independent tasks [**?**].

These mapping constraints restrict PRRs to be homogeneously designed regions, leading to underutilisation of resources. Firstly, after omission of the static area for I/O interconnects, the homogeneous region along the y-axis can be as low as 60% area of the FPGA [**?**]. Secondly, modern data center workloads comprise a range of heterogeneous tasks with varying resource requirements [**?**], [**?**]. Within the boundaries of PRR, the actual area being allocated to heterogeneous task may be lower, namely 38% - 51% [**?**]. Overall, the effect is that the utilisation can be as low as 30% of available resources, resulting in a lower system throughput and reduction in number of tasks that can be co-executed. Finally, due to the routing constraints surrounding fixed PRR slots, the frequency can drop by up to 24% [**?**].

Space sharing optimisations Research has explored optimisation of resource utilisation of PRR-based designs to maximise throughput. Authors in [**?**] design variable sized slots and map a compute intensive task with a memory intensive task. Design in [**?**] allows 1,2 or 4 PRR slots to be used as a reconfigurable region for a task as a mean of mapping options. Authors in [**?**] reduce fragmentation within PRRs by optimising scheduling using policies such as compact placement and minimum conflicts between sharing tasks for variable sized PRRs. Research in [**?**] reduce the reconfiguration overhead associated with PRR usage, by

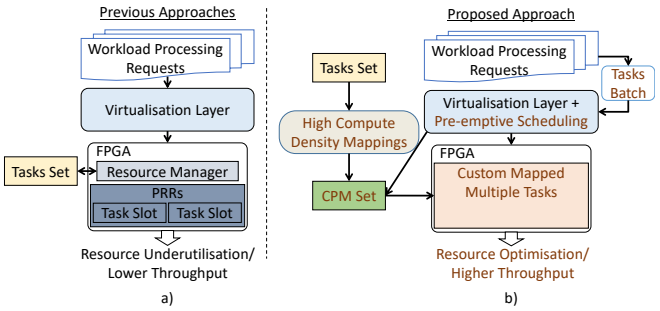

Fig. 2. PRR and CPM design flow and runtime

prefetching the next task's bitstream for next scheduled task and re-using the loaded bitstream when possible. Authors in [**?**] enable runtime support for elastic resource allocation per task if adjacent PRR slots are available, whereas those in [**?**] propose slots built using homogeneous blocks where a task can be allocated any number of blocks at runtime to create heterogeneous slots. However, even though these approaches improve on a basic PRR setup, they use the largely homogeneous PRR slots to map all heterogeneous tasks, which results in non-optimal resource utilisation.

The infrastructure in [**?**] is a recent work that allows creation of amorphous subspaces on the FPGA which can both be partially and independently reconfigured for low-latency mode, whilst multiple subspaces can be combined and custom mapped for high throughput mode. The work shows large gains in throughput by increased resource utilisation when generating combined bitstreams, particularly by co-scheduling tasks with variable resource requirements. However, the work is targeted at the operating system (OS) design and providing system support to transparently transfer between the two modes. For the high throughput mode, it briefly suggests parallel creation of bitstreams (from tasks' netlists) and place and route acceleration techniques to hide bitstream generation latency. However, the proposed use of a fair usage scheduler may create non-optimal mappings and frequent switching at runtime. In contrast, our work looks to first create a much larger design space and systematically explore it offline with the aim of generating higher density designs for high-throughput execution. Furthermore, we incorporate throughput-based scheduling and resource allocation to increase the usage of the high-throughput mode. Finally, we provide a thorough evaluation of both modes to enable informed decisions about changing modes as per the operating environment.

2.1 Design Philosophy

Consider a data computing environment where processing requests are received at regular intervals. The key differences for execution models of PRR and the proposed CPM framework are illustrated in Fig. **??** (a) and (b), respectively. The PRR approach generates bistreams and treats each incoming task requests independently. The system manager manages the incoming task queue via a virtualisation layer that abstracts the underlying hardware implementation from the user space. For each request, the virtualisation layer caters for the I/O and communicates with the resource manager which manages multiple regions and loads the bitstream for each execution.

In our approach, a major focus has been made to improve the resource utilisation by removing the need for each task to go into a predefined partial region, thus improving total execution time for the overall task set. This requires a number of major changes to the flow. Firstly, a range of efficient swappable FPGA accelerators with a high compute density and varying areas are created for each task. The aim is to integrate the bitstreams supporting execution of multiple tasks, but rather than employing dynamic reconfiguration, these are loaded as a single multi-task bitstream onto the FPGA.

A lightweight scheduler, running at the same level as virtualisation layer, links task processing requests with corresponding optimum bitstream based on multi-task DSE and pre-emptive scheduling techniques. To enable intelligent co-scheduling, the scheduler processes task queues in batches, allowing reordering and processing in clusters to gain a higher throughput from CPM. With the order of jobs being second priority, a comparison may be made with other approaches for epoch time at which execution of each task is completed, in order to evaluate the effect of prioritising throughput over the order of tasks.

Furthermore, the tasks implementation and runtime support are provided by the OpenCL framework as OpenCL is now recognised as an established high-level language for design as well as integration with software-based heterogeneous data centres [**?**]. The proposed approach fits well with the FPGA version of cloud computing model of Software as a Service (SaaS) where optimised bitstreams for standard computing tasks are stored as a library (Amazon Marketplace for Amazon FPGA Image (AFI) [**?**]) and users can request functional acceleration service with variable workload. Further optimisations can be made by complementing it with real-time data center workload characterisation [**?**].

We previously presented a comparative evaluation of partitioning schemes [**?**], an accuracy analysis of runtime evaluation tool [**?**] and spatial mapping optimisation via clustering [**?**]. This paper introduces a complete framework and includes the implementation of novel new components for temporal optimisation and integration with high-level virtualisation model. Building on all these modules, this work, for the first time, presents a complete system stack from early design analysis to deployment of dynamic heterogeneous HPC tasks.

3 Methodology

The high compute density mapping of Fig. **??** (b) forms the core of framework enabling task-specific execution model and is explained in Fig. **??**. From the list of *heterogeneous HPC tasks* to be executed, the *OpenCL computing model* is used to describe the functionality, thereby allowing the parallelism granularity to be defined using the general *high-level synthesis parameters*. The resulting *design space* is explored using dynamic hardware profiling, ensuring that the generated designs provide optimum speedup per resources utilisation. The eventual *multiple hardware designs* are then used to generate the dataset specifying the throughput achieved against resources utilised for each task.

The comprehensive *multi-task runtime evaluation tool* starts with the single task DSE to gauge the performance of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

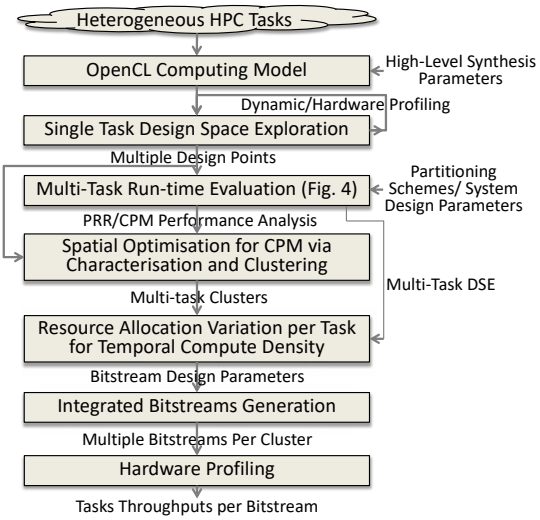

Fig. 3. High compute density mapping design methodology

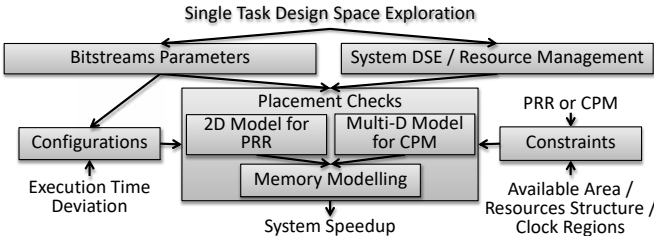

Fig. 4. Runtime evaluation methodology

various *partitioning schemes* for dynamic workloads against various *system design parameters*. The *performance analysis* showed that although CPM provides higher spatial compute density than PRR, it is similar to SAC and may perform worse for dynamic workloads because of lower temporal utilisation. To address this, *characterisation* of tasks using the *single task DSE* along with machine learning based regression models is used to evaluate the weighted relationship between each on- and off-chip heterogeneous resource and FPGA throughput. The characterisation is used to divide all tasks into smaller *clusters*, so that tasks in a cluster complement others' resource needs and can be co-executed on the FPGA.

Furthermore, to minimise the reconfiguration overhead, a multi-task bitstream may only be replaced with a new one after all the tasks being co-executed have finished. To reduce stalling by the longest running task, *resource allocation per task* is varied in the *multi-task cluster* as per the runtime workload size of each co-executed task. To enable this, the module generates a set of designs per cluster using a *multi-task DSE*, while trading off resource allocation (and throughput) between the cluster tasks. This permits variation in the execution time, T_{exec}, of each task in processing respective workloads.

The generated *multi-task designs* are custom mapped on FPGA to *generate bitstreams* supporting multiple task accelerator functions. The generated sets of *bitstreams per cluster* are then *profiled on the hardware* for throughput using metrics defined per corresponding workload for each task in the cluster. These high compute density mappings enable a high throughput hardware design which are then used by the higher level system stack.

3.1 Design Space Exploration

The DSE enables exploration of system optimisation strategies and the resulting area-throughput rate supports a benefit based approach where tasks can be allocated resources which benefits them the most, i.e., memory, compute. The DSE is enabled by OpenCL's capability to allow explicit description of parallel computing and scaling of hardware resources via multiple parameters [**?**]. A task's *kernel* can be scaled over multiple compute units, *CUs*, where these implement coarse-grained parallelism described as work-groups. A work-group can be further spanned over work-items where multiple pipelines for these can be defined via the single instruction multiple data, *SIMD*, pragma.

A kernel may also be implemented as a single work-item. For some of these tasks, task-specific parameters such as the block size, number of rows, are used, as these define the parallelism of the defined parameter size. For some tasks, unrolling pragma, *U*, can be used to *unroll* compute intensive part of the kernel which is identified using *dynamic profiling* based on an 'always active' counter. The counter is coded in VHDL and passed to the OpenCL kernel as a software library via an Intel OpenCL Library feature. Eventually, parameters that provide the highest throughput scaling per unit area are selected.

3.2 Runtime Evaluation

A comprehensive multi-task runtime functional emulation tool (Fig. **??**) allows fast early stage comparison. It enables multi-task DSE using the single task DSE and may be combined with analytical models [**?**], to significantly speed up clustering and resource allocation per task.

Placement Checks: For PRR, the 2D area model treats mapping as a rectangle fitting problem and aims to find a region homogeneous in both size and spatial distribution of resources to map each incoming task [**?**]. For CPM, we implement a multi-dimensional model accommodating a dimension for each heterogeneous, on-chip resource i.e. logic, block random access memory (BRAM), digital signal processing blocks (DSPs). The mapping optimisations try to accommodate as many tasks as possible, whilst keeping total utilisation of all resources within the device limit. For PRR, each task's configuration is treated independently while for CPM, a configuration waits for the longest running task and then selects a new multi-task configuration.

Memory Modelling: Even if the allocated on-chip resources in the multi-task environment are same as the single task, the achieved throughput may not be identical due to memory contention. To model and predict memory performance in multi-task processing, the tool employs ridge regression [**?**]. We generate a range of multi-task bitstreams and measure the actual performance to train the model with the accuracy presented in [**?**].

System DSE/Resource Management: The tool can be used to evaluate various resource management strategies. For CPM, a multi-task DSE estimates the effect on throughput while varying resource allocation per task in a cluster. For PRR, the tool implements optimisations that target segmentation (vacant regions on the FPGA at runtime) on

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

top of basic homogeneous PRRs and which are important to compare PRR fairly with CPM. Among these, the first one checks if the adjacent PRR regions are free, and then attempts to fit a larger bitstream for the same task in this combined region to gain a $speedup$ [**?**].

The second one targets partitioning FPGA into heterogeneous PRRs with different numbers of resources to increase mapping flexibility [**?**]. 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 the current scenario, the device size is not big enough to benefit from such an approach, so heterogeneous PRRs are defined by varying the number of each type of resources while their relative ratios remain the same (Fig. **??**). The optimisation uses heterogeneous PRRs to fit a smaller bitstream for tasks when none of the original bitstreams can be accommodated by a region [**?**].

Finally, the tool varies the vertical step size up to a single row on a continuous y-axis, for bitstream relocation. Although exhaustive, this can be achieved by generating multiple bitstreams equal to the number of $rows$ within each clock region, by varying starting y-coordinates.

Configurations: Bitstreams parameters such as the coordinates of bounding boxes and heterogeneous resources usage from the DSE, are passed to the evaluation tool. The user can also specify deviation in T_{exec} of tasks. A uniform distribution is used for random task generation and an input parameter varies the range of distribution for T_{exec}.

Constraints: In PRR, the homogeneous/heterogeneous regions are fixed and the coordinates are provided by the user. In CPM, the total number of available heterogeneous resources and a realistic percentage of the maximum utilisation is provided as an input. To study the effect of various PRR constraints, the available area for the task mapping as well as bitstream relocation steps can be varied.

3.3 High Compute Density Bitstream Generation

We optimise system throughput by achieving a denser mapping of FPGA resources. The various modules include:

Characterisation and Clustering for Spatial Optimisation: Along with clustering, informed runtime decisions are enabled on heterogeneous resource allocation per task which requires characterisation of the DSE of each task rather than of any single bitstream. We perform regression modelling to find the weighted contribution of each type of resource towards throughput. This allows a benefit-based approach where a task which profits the most from a higher allocation of a certain resource type is clustered with a task which profits the least. Four different resources, including on-chip BRAM, DSPs, logic and off-chip bandwidth which represent the key FPGA resources, are considered.

Whilst ordinary least squares is one of the more commonly used linear regression methods, it is highly sensitive to random errors when variables are correlated, such as here where DSP blocks are linked to BRAM; *ridge regression* avoids this. The normalised values of on- and off-chip heterogeneous resources against maximum available for all bitstreams per task form the independent variables for the regression. The achieved throughput, measured on the actual hardware and normalised to maximum achievable for each task, becomes the dependent variable.

Regression provides the significance scores of each type of resource for each task to scale throughput. In addition, the normalised bandwidth utilisation for each task's largest bitstream is used to cluster tasks for space sharing at a single time. Only bandwidth is used, as unlike other on-chip resources, it may become a bottleneck in the DSE for extremely memory intensive tasks and in the ridge regression model, consistent bandwidth usage may hide the fact that task has a high bandwidth dependence. In order to define clusters, each task is first represented in a multi-dimensional space where each dimension either represents regression score of a resource or normalised bandwidth.

Finding the best combination of clusters is a global optimisation problem. We use a custom-designed optimisation function for its reduced complexity and validity for the considered scale of problem. It runs a set number of iterations where each iteration randomly selects the first task for each new cluster and then other tasks are searched such that they have maximum distance, and thus heterogeneity between them in the multi-dimensional space. The number of tasks in a cluster are defined manually by the system designer based on device size. The sum of mutual distances between tasks is used as a score of the cluster, while sum of all cluster scores defines iteration's score. The iteration with highest score is chosen as the solution.

Resource Variation Per Task: Although CPM allows for higher spatial compute density, all cluster tasks are reconfigured as a single integrated bitstream. This means that the longest running task stalls the other tasks, unless reconfigured at the expense of reconfiguration overhead, and the resource utilisation by the tasks that finish early is suboptimal. To counter this, sets of multiple bitstreams per cluster with varying on-chip resources are created using the multi-task DSE enabled by the runtime evaluation tool and varying the high-level parameters. The clustering helps to avoid contention for the same type of resource and each task is allocated the resources it needs. A range of designs are produced with varying area-throughput rates for processing the respective workloads.

Integrated Bitsreams Generation: A PRR system is limited to pre-defined reconfigurable regions because of the runtime bitstream relocation needs, thus limiting the optimisation via clustering to only off-chip memory bandwidth. However, CPM can also benefit from optimisation of on-chip resource usage using custom FPGA mapping and resource allocation to tasks as per their heterogeneity, to generate a single bitstream offering multiple task acceleration functions.

The bitstream is generated by first using the OpenCL front-end to create HDL modules. Placement scripts modify the constraint files to map task modules to the corresponding PRR while for CPM, the area to be mapped is set to available area for task logic (Fig. **??**). Finally an integrated bitstream is generated using the place and route tools, which are integrated with OpenCL back-end. In this work, no custom mapping optimisations over the vendor's place and route tools are used by CPM for multiple modules. Both the PRR and CPM modules can be partially reconfigured independently of the static logic.

Hardware Profiling: All configurations are profiled while executing all tasks in the cluster to calculate real

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

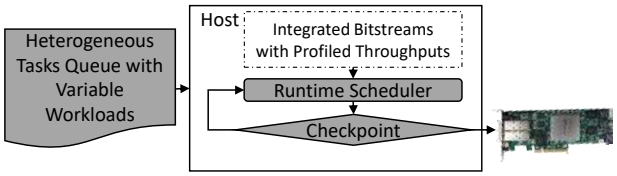

Fig. 5. Runtime scheduler for bitstream selection as per workload sizes

throughput for each bitstream of all tasks using workload-specific metrics, as described in Table **??**. The workload-specific metrics encompass all of the compute, memory and control instructions to give a real measure of performance for time to solution.

With FPGA bitstream generation times taking hours for a single task, the full process can take a significant time. The offline process leading to bitstream generation for an initial set of tasks may be generated at the design time. For upcoming new tasks at runtime and if the system manager decides to update clustering and associated bitstreams, this may be achieved in parallel with tasks execution and updated once completed. Furthermore, the scope and frequency of clustering based optimisation may be limited to reduce design time as well as use of the simulator for faster pre-design analysis.

3.4 Runtime Scheduler

The runtime is designed to be lightweight such that its overhead on task execution time is negligible. For an incoming heterogeneous task queue with variable workload sizes, the runtime scheduler (Fig. **??**) uses preemptive scheduling on the measured throughput of various multi-task bitstreams and selects the one that minimises the difference in the tasks T_{exec}. Meanwhile, checkpoints allow for re-evaluation of the scheduling decisions and context switches. Eventually, this reduces stalling by a single long running task and improves temporal resource utilisation on top of high compute density mappings to achieve higher overall throughput.

Algorithm 1 provides pseudo-code for the runtime selection of optimum bitstream. For each unscheduled task, *nextTask*, in the *task queue*, the algorithm first checks if the other tasks in the same cluster as *nextTask* also need to be scheduled. If not, it uses a single task bitstream for *nextTask*. Otherwise, it iterates through multiple bitstreams for that cluster to find the bitstream that minimises the difference of T_{exec} (calculated as data size divided by throughput) for the tasks in the cluster for their respective workload sizes. The optimum bitstream and corresponding tasks are put in *scheduling order*. The $estimate\_T_i$ value provides estimated T_{exec} for the i_{th} bitstream for task in the brackets. *Max_value* on line 10 equals the sum of T_{exec} for all tasks in SAC.

The preemptive estimation selects the bitstream such that the estimated difference in T_{exec} for all of tasks in a cluster is the least amongst all bitstreams. This does not need cycle accurate estimation and even with allowable difference in the T_{exec} of tasks, a higher throughput is possible, owing to the intelligent clustering and CPM and as long as the longest running task has the highest resource allocation in the cluster.

The runtime also makes use of lightweight scheduling decisions to re-evaluate the bitstream selection or perform

Algorithm 1: Runtime to generate scheduling order, SO, and associated bitstreams, B, for dynamic task queue, TQ

```
1  while TQ is not empty do
2      nextTask ← TQ[nextUnfinishedTask];
3      cluster ← taskClusters[nextTask] ;
4      otherTasksInCluster ← cluster - nextTask ;
5      if otherTasksInCluster not in TQ(unfinished) then
6          B[nextTask] ← singleBitstream[nextTask];
7          SO[nextTask] ← nextTask
8      end
9      else
10         ΔT_exec ← max_value ;
11         for i in range(bitstreamsSet[cluster]) do
12             temp_ΔT_exec ← estimate_T_i(nextTask) -
                estimate_T_i(otherTasksInCluster) ;
13             if ΔT_exec > temp_ΔT_exec) then
14                 ΔT_exec ← temp_ΔT_exec ;
15                 B[nextTask] ← bitstreams[i] ;
16                 SO[nextTask] ← nextTask +
                    otherTasksInCluster;
17             end
18         end
19     end
20 end
```

context switches at discrete points in time, called *checkpoints*. Although checkpoints can be implemented at a finer granularity in OpenCL, such as at work-group level [**?**], the reconfiguration and host-accelerator communication overhead, may offset the acceleration gain. Instead, checkpoints are implemented at *block* level, namely an independent part of a whole workload that is batch processed, i.e., the computation is off-loaded to a FPGA and outputs written back to the host in batch. A block may be workload specific and is generated via workload distribution at a higher software level. All OpenCL workloads need to be processed as a set of blocks, iterated over the total workload, due to limited on-board DRAM.

Considering the runtime workload, the scheduler uses the preemptive estimation of T_{exec} at the respective block size and the number of blocks in order to estimate total T_{exec} of a workload. However, at the checkpoints after each block execution, the bitstream selection and scheduling decisions are re-evaluated, allowing for variation in the resource allocation per task. The scheduler evaluates the following equation to decide on when to switch the bitstream:

$$New\,T_{exec} = T_r(n) + \frac{W_S(r)}{T_H(n)} \tag{1}$$

where $T_r(n)$ and $T_H(n)$ are the reconfiguration time and throughput of the new bitstream, respectively, while $W_S(r)$ is the remaining workload. The scheduler compares the $New\,T_{exec}$ with the current configuration and chooses the improved one. Recent work in this domain has proposed the design of mechanisms for context-saving as well as runtime migration to a new reconfiguration on the same device or a new device for OpenCL-based tasks while saving the already processed data[**?**], [**?**].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

TABLE 1
Use Cases Characteristics.. Compute Units (CU) scaling is always for the whole kernel. U is for Unrolling and BW for bandwidth

Use Case	Scaling Parameters	Computing Characteristics	Throughput Metric / S	# Designs	Speedup
Page Rank (PR) [?]	CUs and U new rank calculation for each page	Inefficient pipeline implementation	# Links	9	6×
Alternative Least Square (ALS) [?]	CUs and U error calculation in recommendation estimation	Severely memory latency bound	# Users per unit Items	4	2×
Binomial Option Pricing (BOP) [?]	CUs and U Binomial Tree traversal	Highly compute and on-chip memory bound	# Options	8	21×
Breadth First Search (BFS) [?]	U edges traversal of a single node	On-chip memory and BW	# Edges	5	5×
Sparse Matrix Vector multiplication (SpMV) [?]	U sum calculation of each row	Logic, on-chip memory and BW	# Non-zeros	6	190×
Finite Difference Time Domain (FDTD) [?]	# Points in a sliding window processed in parallel	High dependency on all resources	# Points	5	13×
Lower Upper Decomposition (LUD) [?]	CUs and U compute intensive loops in decomposition	Compute and BW bound	# FLOPs	7	18×
Video Decomposition (VD) [?]	# rows of pixels being processed in parallel	On-chip resources for bigger and BW for smaller designs	# Pixels	6	8×
Matrix Matrix multiplication (MM) [?]	SIMD pragma for complete pipeline and U calculation of dot product	High dependency on all resources	# FLOPs	8	204×
Needleman-Wunsch (NW) [?]	Size of strings to divide the problem and are processed in parallel	High dependency on all resources	Sequences Length	7	33×
K-nearest Neighbour (NN) [?]	U distance calculation to other points	Highly efficient pipeline requiring high BW	# Points	4	5×

3.5 Virtualisation and Revenue Model

The conventional PRR-based approach targets infrastructure as a service (IaaS) and scales area per tasks with the number of PRR slots corresponding to number of resources [?], similar to the discrete cores-based distribution of resources in multi-processor systems. As resource-based division incurs a high cost of sharing in FPGA, a task-based functional acceleration stack targeting SaaS model is employed where functionality is offered to users as a service while the area scales with the associated throughput. This is similar to the Amazon model which provides AFIs containing pre-synthesised bitstreams of FPGA functionality while hiding the implementation details from users [?]. With CPM, the model can be extended to area shared multi-task execution with a higher system throughput facilitating higher revenue.

To enable the task-based model, we use the VineTalk virtualisation framework [?]. VineTalk reduces the efforts needed by application developers to deploy FPGA-based acceleration in data centres by handling the communication and control between an application and underlying accelerator. It does that by registering tasks as a set of supported functionalities accessible via easy to use interfaces while hiding the underlying hardware implementation details. It essentially exposes FPGA as a virtual accelerator, *VineAccelerator*, available to applications as task-based software API. The underlying libraries then manage the task queues and data buffers. Decoupled from this is the *software controller* and hardware facing API, which communicates with the underlying FPGA's vendor runtime and manage bitstream loading and host-accelerator data transfers associated with each VineAccelerator.

We modify various layers to incorporate Vinetalk for area-shared CPM multi-task processing. At the application level, multiple tasks in a cluster can be integrated into a single call associated with a single VineAccelerator. In other words, each VineAccelerator represents a unique cluster and its definition then manages execution of independent tasks in a cluster. Furthermore, each VineAccelerator has access to multiple bitstreams, where each bitstream represents varying throughputs for each task in the cluster via an associated set of parameters. These are used by the scheduler to select the optimum bitstream at runtime and passed to the software controller. Finally, we implement interfaces and drivers for the hardware facing API for integration with Intel FPGAs.

4 Test Environment

We considered 11 HPC tasks belonging to various application domains and computing dwarfs such as sparse and dense linear algebra, graph analytics, structured grid computing and dynamic programming. We then identify high-level parameters for each task for DSE for OpenCL implementation. The parameters are evaluated to provide the maximum throughput per resource usage using hardware profiling. We also identify the key characteristics identified after profiling, that project a need for task characterisation on actual hardware and verify the selection of tasks for comprehensive evaluation. The DSE along with workload specific metrics used for each task in the throughput calculation, are summarised in Table **??**.

4.1 FPGA and Host Platform

The runtime evaluation tool is written in Python 3. The high-level DSE is performed via the Intel OpenCL SDK for FPGAs v16.1 while constrained placement is achieved using Quartus Prime v16.1. The hardware profiling is performed for the Nallatech 385 board as the target FPGA system. The power is measured using on-board power sensors accessible via the memory-mapped device layer whilst the bandwidth is measured using the Intel FPGA Dynamic Profiler for OpenCL GUI. OpenCL runtime and independent *command queues* are used for each task, allowing parallel execution. Within a single command queue, non-blocking calls are issued for memory transfers as well as to multiple kernels of a single task, if needed. The runtime is executed on *host* comprising of an Intel Xeon E5530 chip running at 2.4 GHz.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

4.2 System Throughput Metric

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 task's throughput does not provide an indication of system performance; the contribution to absolute processing time and total speedup may be influenced more by the tasks with larger workload sizes. Generic metrics such as FLOPS etc. may not provide a meaningful measure for all of the tasks being evaluated.

We use two different metrics for emulated and hardware results in order to allow a realistic and comprehensive assessment to be made. Firstly, the emulation of large task queues comprising range of tasks provides the potential to estimate the total $speedup$, measured as execution time to process the whole workload or task queue for various partitioning schemes and SAC. To evaluate the compute density provided by various approaches in a multi-task environment, the STP metric [**?**] used is defined by:

$$STP = \sum_{i=1}^{n} NP_i = \sum_{i=1}^{n} \frac{C_i^{SP}}{C_i^{MP}} \tag{2}$$

where NP is each task's normalised progress defined by the number of clock cycles it takes in single task mode, C_i^{SP}, when the task has all of the resources of the FPGA available 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. The metric encompasses various system design parameters such as throughput variation with resource allocation per task, compute density variation against resource utilisation and system performance (including STP/Watt for energy efficiency) for various space partitioning schemes. It then provides a throughput relative to a baseline of SAC, which has STP value of 1.

5 Results And Analysis

The scope of DSE is first explored with emulated results before analysis using clustering and runtime performance variation with design parameters. Whilst the results target the low-level system performance, we discuss the high-level virtualisation model using CPM.

5.1 Design Space Exploration

The DSE provides real area numbers as well as variation in throughput against resource utilisation for fair evaluation and comparison of mapping schemes. The scaled parameters and achieved speedup are summarised in Table **??**. To measure speedup, the baseline T_{exec}, corresponding to the lowest area bitstream, is defined by the serial pipelined benchmark implementation of the task. The maximum throughput is defined by the largest bitstream, limited by FPGA resources. We have generated $4-9$ designs per task where each represents a point on the area-throughput curve and the maximum speedup ranges from $2\times$ to $204\times$ for tasks being considered.

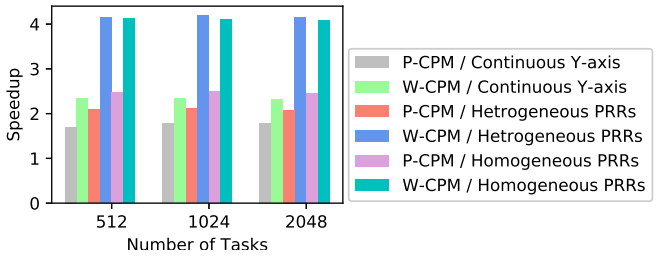

Fig. 6. $Speedup$ achieved by CPM versus PRR mapping

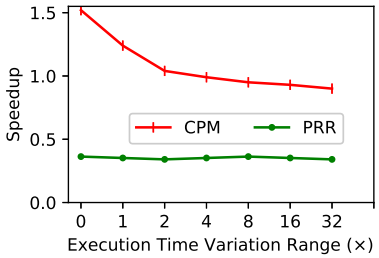

Fig. 7. $Speedup$ variation with variation in execution times (Tasks = 1024)

5.2 PRR vs CPM

The evaluation tool provides the projected runtime gains of CPM against SAC and various PRR strategies. The tool also allows variation of system parameters and task constraints that affect the CPM throughput, particularly due to lower independence in time as compared to PRR and SAC. We iteratively apply different constraints to distinguish and highlight their effects while using all of the considered 11 heterogeneous tasks but with varying workloads.

Maximum theoretical gain by CPM: Firstly, the maximum theoretical $speedup$ achieved by CPM against various types of the PRR mapping, namely the continuous y-axis, heterogeneous PRRs and homogeneous PRRs, are analysed using runtime evaluation tool. Here, we consider an ideal scenario for CPM where tasks sharing the space have same execution times. In total, there are 80 $rows$ of the FPGA that can be configured as a single region or a set of two homogeneous regions of 40 $rows$ each. Two more heterogeneous PRRs, namely 30 and 50 $rows$, are defined based on the sizes of generated bitstreams.

For CPM, we either use the same region as used for the PRR to maintain homogeneity, (*Partial CPM*) (P-CPM), or use all of the available area for the task logic after placement of the static modules, (*Whole CPM*) (W-CPM). The designs for W-CPM and P-CPM are created by using appropriate scaling parameters as well as the placement constraints for each realisation as well as the placement constraints. Due to lower area utilisation, P-CPM takes lower bitstream generation time and has slightly lower reconfiguration overhead. This analysis 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 in a custom fashion.

Fig. **??** shows that for an ideal environment for CPM, it can achieve up to $4.1\times$ higher throughput as compared to PRR, measured in terms of the total execution time for a set task queue size. Please note that out of this $4.1\times$ gain, a $2\times$ $speedup$ is achieved via heterogeneous custom mapping whilst the rest is achieved by exploiting the higher resource availability. The results show that if the y-axis can be made

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

continuous, a throughput gain of $1.8\times$ can be achieved while heterogeneous PRRs can improve performance while making use of various optimisations mentioned in Section **??**.

T_{exec} **variation:** The *speedup* reported in Fig. **??** considers an ideal scenario for CPM by using similar T_{exec} for all tasks sharing the FPGA at any time, however, this is not the case in dynamic environments. Next, the relative T_{exec} of the tasks is varied, with reconfiguration only after all co-executed tasks have finished processing, thus allowing analysis of its effect on *speedup*. The speedup is given against baseline of SAC for both partitioning schemes.

The results in Fig. **??** depict a surprising trend, particularly for CPM versus PRR. Even with increasing range of T_{exec} by up to $32\times$ (beyond this range a reconfiguration overhead would become negligible for most tasks), the *speedup* with CPM decreases but remains higher than PRR by $2.7\times$. This is because on average, the device may be used by 3 or less tasks using CPM, as constrained by the size of the FPGA chip. Thus, a task may stall up to 2 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 is not the smallest. Against SAC, the CPM follows a similar trend, however, the speedup falls below 1 as the range of T_{exec} goes higher than 2 whilst PRR maintains an average speedup of $0.37\times$.

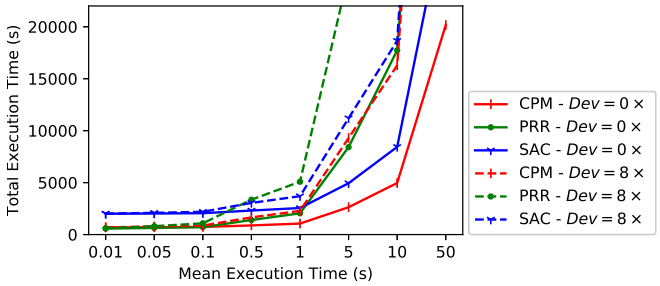

Fig. 8. Total execution time including the reconfiguration overhead for varying mean of tasks individual execution time (Tasks = 1024

Reconfiguration overhead: The analysis so far has not considered reconfiguration overhead, which can be significant if the tasks to be executed have smaller workloads in more dynamic environments. 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 CPM and SAC, the gains may be offset by the higher reconfiguration overhead.

The next set of experiments evaluate the effect of reconfiguration overhead on total T_{exec} to process a task queue against varying mean and range of T_{exec} of tasks. The experiments consider reconfiguration time for each scheme as proportional to the area to be reconfigured. The results are shown in Fig. **??** and include two different ranges (R) for each evaluated mean. Starting from the offset of reconfiguration overhead, the total T_{exec} generally increases linearly with increase in mean T_{exec} of tasks. However, the lower reconfiguration overhead plays more significant role towards better performance for smaller tasks with lower mean T_{exec} while higher throughput is more significant for larger tasks.

In the first test scenario, all tasks have similar T_{exec} i.e range approaches 0 which, as we mentioned earlier, is an ideal scenario for CPM. However, even with the lowest throughput associated with PRR, it provides the best overall system performance by up to $1.2\times$ owing to the lowest reconfiguration overhead. The lower throughput for PRR becomes the more significant factor towards total T_{exec} 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 $8\times$ range of T_{exec}. Without considering reconfiguration overhead, CPM performed worse than SAC (Fig. **??**). However, lower number of resources per task results in lower reconfiguration overhead for CPM as compared to SAC. This results in CPM providing better overall performance by up to $1.14\times$ even at mean T_{exec} of 10s. The overhead only offsets the performance loss due to lower throughput though, as CPM may perform worse than SAC with even higher range.

Although this work is focussed on identifying the use of CPM for higher throughput and optimising it for dynamic environments, the detailed comparison against various constraints in this section highlights the need for analysis of various schemes to suit the operating environment. Such analysis may also enable a system to switch mapping schemes at runtime as per variation in task dynamics, as proposed in [**?**].

As for the CPM, although it allows a space-shared model allowing scaling of resources at sub-device level, the degradation of performance against SAC suggested the need for further optimisations. A quick analysis also showed that apart from the underutilisation of resources in time by tasks with variable T_{exec}, the runtime spatial utilisation by heterogeneous tasks is limited to 62%, 49% and 71% on average for logic, BRAM and DSPs, respectively.

5.3 High Compute Density Mappings

To analyse the gains made by the proposed approaches for high density mappings, we first establish a baseline system throughput before analysing various optimisations.

5.3.1 Baseline System Throughput for CPM and PRR

The size of the device being used is small, whilst further constraints on area available are placed on it by PRR. This limits the area sharing to a cluster of 2 tasks. For CPM, up to 3 tasks can be accommodated at one time. However, based on practicalities and the need to keep the comparison fair, it makes sense to use 2 tasks per cluster for CPM as well. Using the DSE, the largest bitstreams per task are selected within the area constraints of the PRR and CPM.

We generate 10 random clusters of tasks for both PRR and CPM as baseline. For evaluation of the STP, the data sizes for tasks in a cluster are chosen such that both tasks have a similar T_{exec}. The results in Fig. **??** (a) show that CPM with an average STP of 0.99 can provide an average $2.4\times$ higher throughput as compared to PRR's STP of 0.41 on the basis of a higher compute density in space but with consuming higher power. Fig. **??** (a) shows that although the gain is less, the CPM provides $1.9\times$ better energy efficiency (STP/W) on average over PRR.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

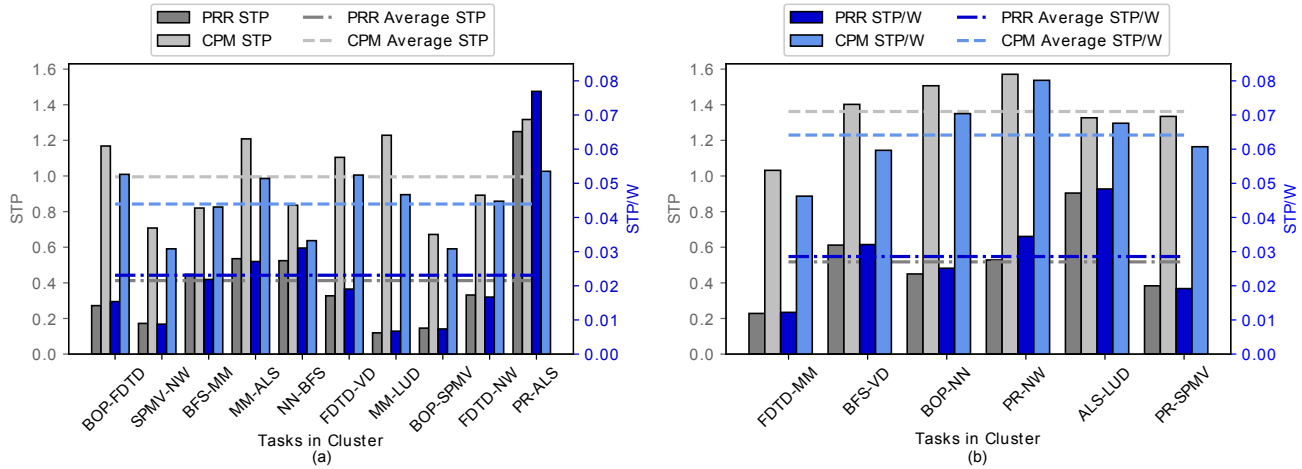

Fig. 9. STP and STP/W for CPM and PRR using: (a) unoptimised clustering, and (b) optimised clustering

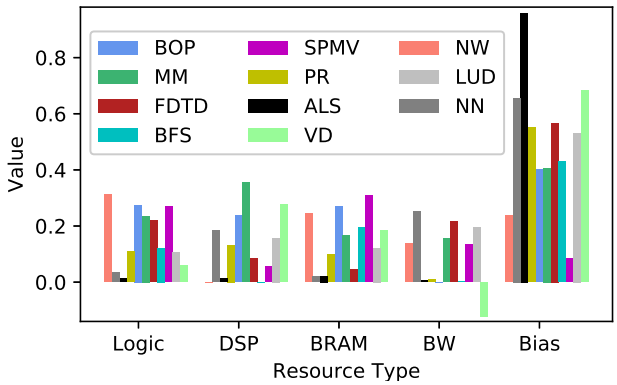

Fig. 10. Ridge Regression models for HPC tasks

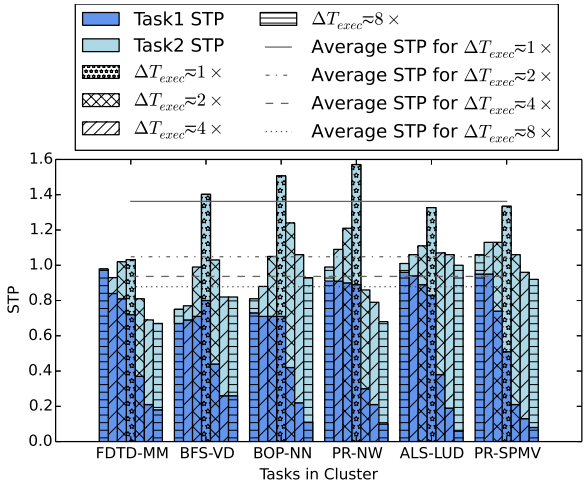

Fig. 11. STP variation for CPM with variation in T_{exec} of tasks in cluster

5.3.2 *Clustering*

Before describing the improved throughput due to clustering in Fig. **??** (b), we briefly show the contribution of resources towards tasks' throughput using the DSE and Ridge Regression in Fig. **??**, which forms the basis for clustering. The bias value in the figure representing the constant in the modelled linear equations relates to the baseline throughput. The clustering algorithm uses these models to create a set of 6 optimum clusters.

Fig. **??** (b) shows gains for both the PRR and CPM, using a similar T_{exec} for the tasks in a cluster. For PRR, the STP increase of $1.2\times$ to 0.5 is mostly due to optimisation of the off-chip memory bandwidth utilisation. For CPM, the STP increase of $1.4\times$ to new value of 1.4 corresponds to both the on- and off-chip resource optimisation. The gain of $3.3\times$ and $2.8\times$ for the throughput (STP) and energy efficiency (STP/W) between the CPM in Fig. **??** (b) and the PRR in Fig. **??** (a) respectively, is the maximum achievable via the proposed optimisations as compared to the existing area-shared schemes while an STP gain of $1.4\times$ compared to SAC is achieved.

5.4 T_{exec} Variation

Until now, the experiments have used custom data sizes which ensured a similar T_{exec} for co-executed tasks which may represent a static configuration for long running tasks. However, for a more dynamic task queue, the limitation of CPM is that the execution is stalled by the longest running task, unless the scheduler decides to reconfigure the FPGA. In the next set of experiments, the T_{exec} of tasks is varied relative to each other. The results are shown in Fig. **??** where the δT_{exec} represents difference in T_{exec} for tasks in a cluster as a multiple.

For each sample cluster, the middle cluster represents similar T_{exec}, that is $\delta T_{exec} \approx 1$. Moving either side, the δT_{exec} increases with left side representing the first task in x-label taking the longer time to execute while the right side represents the second task, as shown by individual STP of tasks. Considering the average values, the results show that STP for CPM drops sharply initially but then stabilises. For example, from $1\times$ to $2\times$, the drop is by $1.3\times$ (from 1.36 to 1.04), however, from $4\times$ to $8\times$, even when variation in T_{exec} is $4\times$ or more, the drop is only $1.06\times$ (0.94 to 0.88).

The reason can be seen from the individual contribution of each task to the total STP. With an increase in T_{exec} variation, the STP is increasingly defined by the longest running task and becomes independent of the variation in T_{exec}. Furthermore, the individual STP contribution by the longest running task improves with increasing the T_{exec} variation as it gets a higher share of the off-chip memory bandwidth. All of these factors reduce the effect of variation in T_{exec}, but it still causes a significant drop in STP by $1.54\times$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

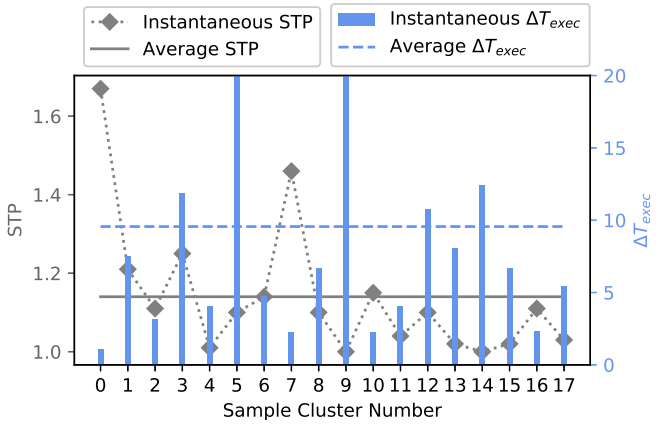

Fig. 12. STP and STP/W for variable workloads via dynamic bitstream selection for 18 clusters that are used for execution of 36 tasks

5.5 Runtime

To counter the drop in the STP with variation in T_{exec}, the framework generates 3-4 bitstreams per cluster, with each bitstream trading off resources for each task against the other. We then evaluate a complete system using a queue of 36 randomly generated task workloads involving 3 workloads per task with variable sizes. The range of workload size per task is such that they take 1 - 60 mins (in line with study of real workloads processing times on three Google clusters in [**?**]) to process using bitstreams from Fig. **??** (b). Each workload comprises of a number of workload-specific blocks such as matrices, image frames, options, graphs, etc. Furthermore, for the sake of the experiments, each request is treated as an independent task and the module reuse strategy to avoid reconfiguration is not considered if the same task has multiple requests. If so, multiple requests for same task can be combined to form a single larger task.

The runtime scheduler then selects the optimum bitstream, that minimises the T_{exec} variation in a cluster, using the profiled workload-specific metrics for throughput and the preemptive scheduling. As mentioned earlier, this estimation only chooses the best fit from the available set and does not require a cycle accurate estimation of T_{exec}. The runtime can also use single task per bitstream (SAC), if needed. For the considered task queue, the scheduler reconfigured the bitstream at checkpoints only once while for 2 of the workloads, it used SAC.

The achieved instantaneous and average STP for 18 bitstreams of 2 tasks each used for execution of 36 tasks (SAC is also shown as a cluster, but represents two different bitstreams), is shown in Fig. **??**. The figure also includes the corresponding T_{exec} variation for bitstreams from Fig. **??** (b), with an average of $9.6\times$. This set of experiments also includes the reconfiguration overhead, however, it is insignificant (less than 2s) for the size of workloads being considered. The results show that using the intelligent runtime selection of the bitstream, the STP can be improved by $1.3\times$ as compared to Fig. **??**, while reducing the power by 5%, on average. The actual processing time or STP for the PRR-based processing for the generated task queue could not be provided due to non-availability of the dynamic reconfiguration framework needed for the tasks with varying T_{exec}. However, the STP would be similar to that reported in Section **??**. The overall achieved STP is $2.8\times$ higher than the base value using PRR in Fig. **??** (a) while being $2.3\times$ more energy efficient.

5.6 Discussion

The focus of this work has been to improve FPGA compute density and hence has not commented on the virtualisation overhead and the data transfer from the host to the FPGA via PCIe. Both the VineTalk and scheduler overhead depends on the workload size and number of batches required to complete processing. For a single batch the combined overhead can be as low as $100\mu s$, while excluding the reconfiguration overhead. For higher number of batches (256), the overhead varies for different tasks and lies in the range of 0.3 - 0.9 % of total execution time of tasks.

Furthermore, the cost of data transfer is essentially the same for all of the partitioning schemes. However, compared to SAC, multi-task processing provides up to $1.2\times$ lower total T_{exec} including the memory transfers from the host. In this, $1.18\times$ is due to the higher compute density and is in the similar range as achieved STP for the set of evaluated workloads. The rest is provided by the time multiplexed memory transfers for multi-task processing, with the longer running task starting execution whilst the data is being transferred for other tasks.

The framework processes the task queue in batches where jobs in the batch may be reordered to maximise throughput. However, the order of jobs is the second priority and thus, the first task to be scheduled is selected from the order and the second task corresponds to the respective cluster. Using the runtime evaluation tool, it showed that although the PRR allows a strict order of processing to be followed, the lower compute density means that 100% tasks finish execution later in terms of clock time compared to CPM. In a multi-FPGA data centre, multiple clusters can also be offloaded to different FPGAs to maintain adherence to the execution order. Tasks with strict deadlines can also be executed in SAC.

STP as a metric defines throughput as a comparison against SAC and the theoretical limit for maximum possible STP is defined by the number of tasks being shared i.e. 2 in this case. The evaluations show that the CPM provides up to a maximum of $1.4\times$ improvement in the throughput which drops to $1.18\times$ for the more dynamic and variable size workloads. This will improve with larger devices which will allow more space sharing than 2 tasks. PRR, however, reduces the STP to $0.5\times$. There may be other benefits to using PRR particularly for more dynamic fine-grained workloads or faster integration of a new functionality. However, STP must be considered for evaluation of system performance and the CPM has the ability to provide a higher system throughput than SAC, while allowing benefits of area sharing. Even for larger workloads requiring multiple FPGAs, multiple instances can be generated on multiple FPGAs where each instance shares space with instances from other tasks to improve resource utilisation. Each instance may be treated as an independent sub-task by the scheduler to optimise temporal usage with changing throughput requirements from the main task.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3101153, IEEE Transactions on Parallel and Distributed Systems

6 Conclusion

A systematic framework is proposed for addressing the challenges of virtualisation based on space sharing of FPGAs and achieving higher system throughput. The framework proposes characterisation and clustering of tasks based on their heterogeneities in resource usage, which is then complemented by custom mapping and partitioning of tasks to maximise utilisation in space. A lightweight runtime scheduler integrated with a higher level virtualisation layer then makes use of off-line profiling of resource allocation variation per task to increase compute density in time. In doing so, the work projects the trade-offs of various space partitioning schemes, while improving the throughput and energy efficiency over existing methods.

References

[1] Amazon, "Amazon EC2 F1 instances," 2018, accessed 1 June 2020. [Online]. Available: https://aws.amazon.com/ec2/

[2] Z.-H. Zhan et al., "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comp. Surveys*, vol. 47, no. 4, 2015.

[3] A. Vaishnav et al., "Resource elastic virtualization for FPGAs using OpenCL," in *FPL*, IEEE, 2018.

[4] M. Asiatici et al., "Virtualized execution runtime for FPGA accelerators in the cloud," *IEEE Access*, vol. 5, pp. 1900–1910, 2017.

[5] X. Changet al., "Effective modeling approach for IaaS data center performance analysis under heterogeneous workload," *IEEE Trans. on Cloud Comp.*, vol. 6, no. 4, pp. 991–1003, 2016.

[6] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE Micro*, vol. 28, no. 3, 2008.

[7] N. Eskandari et al.,"A modular heterogeneous stack for deploying FPGAs and CPUs in the data center," in *FPGA*, ACM 2019.

[8] R. Kirchgessner et al., "Low-overhead FPGA middleware for application portability and productivity," *ACM TRETS*, vol. 8, 2015.

[9] Y. Zha and J. Li, "Virtualizing FPGAs in the cloud," in *ASPLOS*, ACM, 2020.

[10] A. Vaishnav et al., "FOS: A modular FPGA operating system for dynamic workloads," *arXiv preprint arXiv:2001.09990*, 2020.

[11] M. Huang et al., "Programming and runtime support to blaze fpga accelerator deployment at datacenter scale," in *SoCC*, ACM, 2016.

[12] S. Mavridis et al., "Vinetalk: Simplifying software access and sharing of FPGAs in datacenters," in *FPL*, IEEE 2017.

[13] U. Minhas et al., "Nanostreams: A microserver architecture for real-time analytics on fast data streams," *IEEE TMSCS*, 2017.

[14] N. Tarafdar et al., "Enabling flexible network FPGA clusters in a heterogeneous cloud data center," in *FPGA*. ACM, 2017.

[15] H. Liang et al., "Parallelizing hardware tasks on multicontext FPGA with efficient placement and scheduling algorithms," *IEEE Trans. on CAD of Int. Circ. and Syst.*, vol. 37, no. 2, 2018.

[16] K. D. PhamCAD of Int. Circ. and Syst. "Bitman: A tool and api for FPGA bitstream manipulations," in *DATE*. IEEE, 2017.

[17] K. Vipin and S. A. Fahmy, "Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration," in *ARC*, Springer, 2012.

[18] S. Yazdanshenas and V. Betz, "Interconnect solutions for virtualized Field-Programmable Gate Arrays," *IEEE Access*, vol. 6, 2018.

[19] F. Chen et al., "Enabling FPGAs in the cloud," in *Computing Frontiers*, ACM, 2014.

[20] O. Knodel et al., "RC3E: Reconfigurable accelerators in data centres and their provision by adapted service models," in *CLOUD*, IEEE, 2016.

[21] A. Morales-Villanueva et al., "Configuration prefetching and reuse for preemptive hardware multitasking on partially reconfigurable FPGAs," in *DATE*, IEEE, 2016.

[22] A. Khawaja et al., "Sharing, protection, and compatibility for reconfigurable fabric with amorphos," in *OSDI*, USENIX, 2018.

[23] U. I. Minhas et al., "Evaluation of FPGA partitioning schemes for time and space sharing of heterogeneous tasks," in *ARC*, Springer, 2019.

[24] ——, "Work-in-progress: Design space exploration of multi-task processing on space shared FPGAs," in *CODES+ ISSS*, IEEE, 2019.

[25] ——, "Optimisation of system throughput exploiting tasks heterogeneity on space shared FPGAs," in *ICFPT*, IEEE, 2019.

[26] Q. Gautier et al., "Spector: An OpenCL FPGA benchmark suite," in *ICFPT*, IEEE, 2016.

[27] S. Wang et al., "FlexCL: An analytical performance model for OpenCL workloads on flexible FPGAs," in *DAC*, IEEE, 2017.

[28] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, 1970.

[29] G. Charitopoulos et al., "Run-time management of systems with partially reconfigurable FPGAs," *Integration*, vol. 57, pp. 34–44, 2017.

[30] A. Vaishnav et al., "Live migration for OpenCL FPGA accelerators," in *ICFPT*, IEEE, 2018.

[31] Y. Zhou et al., "Large-scale parallel collaborative filtering for the Netflix prize," in *AAIM*, Springer, 2008.

[32] L. Page et al., "The pagerank citation ranking: Bringing order to the web," Tech. Rep., 1998.

[33] U. I. Minhas et al., "Exploring functional acceleration of OpenCL on FPGAs and GPUs through platform-independent optimizations." in *ARC*, Springer, 2018.

[34] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *Symp. on Workload Char.*, IEEE, 2009.

[35] Intel, "Developer zone. intel FPGA SDK for OpenCL," 2018, accessed 1 June 2020. [Online]. Available: https://www.intel.com

[36] M. Schwarzkopf et al., "Omega: flexible, scalable schedulers for large compute clusters," in *Eur. Conf. on Comp. Syst.*, ACM, 2013.

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 use of heterogeneous systems, particularly FPGA based SoCs, for energy efficient computing.

Roger Woods (M95SM01) received the B.Sc and Ph.D. degree from Queens University Belfast, U.K., in 1985 and 1990, respectively and is a Professor and research cluster director at the university. He has also formed Analytics Engines Ltd. and acts as their Chief Scientist. His research interests are in heterogeneous programmable systems and design tools for data, signal and image processing, and telecommunications.

Dimitrios S. Nikolopoulos FBCS FIET Senior Member ACM (M'97-SM'11) earned the B.Sc.(Hons.), MSc and Ph.D. degrees from University of Patras in 1996, 1997 and 2000. He is currently John W. Hancock Professor of Engineering and Professor of Computer Science at Virginia Tech. His research explores how software at the system level can improve the efficiency, performance, and dependability of computing systems.

Georgios Karakonstantis is a Senior Lecturer (Associate Professor) at Queens University Belfast leading the efforts on error-resillient design in the ECIT Institute. He has published more than 75 papers in peer reviewed IEEE and ACM journals and conferences, and is inventor of a US patent. His research focuses on energy-efficient and error-resilient computing and storage architectures for embedded and high-performance applications.