Lattice-based Scheduling for Multi-FPGA Systems

Teng Yu*, Bo Feng‡, Mark Stillwell†, Liucheng Guo†, Yuchun Ma‡, John Thomson*
*University of St Andrews, UK †Imperial College London, UK ‡Tsinghua University, China
Corresponding Email: j.thomson@st-andrews.ac.uk

***Abstract*—Accelerators are becoming increasingly prevalent in distributed computation. FPGAs have been shown to be fast and power efficient for particular tasks, yet scheduling on FPGA-based multi-accelerator systems is challenging when workloads vary significantly in granularity in terms of task size and/or number of computational units required. We present a novel approach for dynamically scheduling tasks on networked multi-FPGA systems which maintains high performance, even in the presence of irregular tasks. Our topological ranking-based scheduling allows realistic irregular workloads to be processed while maintaining a significantly higher level of performance than existing schedulers.**

***Index Terms*—runtime scheduling, lattice representation, multi-FPGA**

I. Introduction

FPGA-based accelerators have been shown to perform exceptionally well in terms of parallel performance and energy efficiency, yet the generality of this approach is a significant issue in a distributed multi-accelerator environments. A key challenge for these platforms is to maintain high levels of performance on diverse workloads where task size and computational elements requirements vary dynamically. In this paper we present a topological ranking-based scheduling approach which significantly outperforms both the existing scheduler $Orchestrator$ used in the Maxeler DataFlow Engine cluster [1], but also other scheduling approaches [2], [3] targeted at other systems problems. The use of multiple accelerators in distributed computation is becoming increasingly important both in industry and research communities [4], [5]. Dynamically scheduling irregular workloads in a FPGA-based multi-accelerator system is a complex task. To find an adequate solution, *we need to find a way to efficiently schedule multiple irregular tasks and assign corresponding resources to hardware accelerators during runtime*. This can be difficult to achieve in practice for a number of reasons: (1) Multiple tasks should be executed quickly and simultaneously to achieve a short total execution time and high throughput; (2) tasks can be instantiated on different system configurations with differing numbers of accelerators, so an ideal configuration can not be easily determined a priori; (3) tasks require different topologies between hardware accelerators based on what communication is needed during runtime. ***Motivating Example:*** Consider the abstract case as shown in Fig. 1 – a cluster containing two FPGA-based multi-accelerator devices, each of which has eight interconnected accelerators in a state where some accelerators are busy. Three new tasks are dynamically sent to the system with different resource requirements and connectivity constraints. Using a

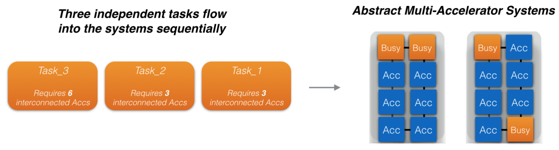
Three independent tasks flow into the systems sequentially
Abstract Multi-Accelerator Systems
Task_3
Requires 6 interconnected Accs
Task_2
Requires 3 interconnected Accs
Task_1
Requires 3 interconnected Accs
Busy
Busy
Acc
Busy
Acc
Busy

Fig. 1. Abstract case of multi-task runtime scheduling on multi-FPGA systems

First-In-First-Out (FIFO) approach, the first two tasks will be assigned to the first device as there is vacancy, but the third task cannot be allocated to any device as it requires six interconnected accelerators. This means the scheduling will stall until new resources are released by the completion of previous tasks. It is clear that current $load$-$balancing$ based heuristics [2], which would allocate Tasks 1 and 2 to different accelerator clusters, also fail to perform in this scenario. Similarly, *Maximum-execution-time* based heuristics [3] do not necessarily help here if Task 3 does not have the longest execution time although it asks for more resources – a common occurrence. This challenge leads to the main motivation of the work presented in this paper: to give a generalised method to design high-performance runtime scheduling methods in multi-FPGA systems targeting irregular tasks with arbitrary capacity representations. Different tasks can ask for different size and topology of accelerators: large workloads may need more accelerators to satisfy timing requirements, and tasks that are capable of exploiting multiple accelerators and specific topologies should have potential access to such resources.

This work proposes a dynamic scheduling system which efficiently schedules irregular workloads using a novel heuristic-based runtime resource allocation methodology. It uses a representation which is general enough to be used across different types of accelerator system, and is capable of representing the required resource and connectivity requirements for workloads. Leveraging this representation, we build a ranking-based scheduler.

II. Background and Related Work

First-In-First-Out (FIFO) based strategies are most commonly used in current multi-accelerator systems to schedule tasks for both research and industry, such as PALMOS [6] for GPU-based systems and AMC [7], PPMC [8] for FPGA-based systems, and commercial Maxeler Orchestrator [1] customized for Xilinx FPGA-based devices in Maxeler Technologies. The initial order of tasks is used as a basic heuristic to guide the scheduler which avoid making any comparison to achieve quick decisions. All of these works show the FIFO approach works well in homogeneous cases where tasks are simple and similar. We use FIFO as one baseline in our evaluation. More sophisticated approaches have been proposed, such as the Heterogeneous Earliest Finish Time (HEFT) algorithm targeting on heterogeneous processors [3] and MFIT algorithm targeting on multi-FPGA systems [2]. HEFT shows satisfactory performance in both quality and cost of schedules in DAG-based task scheduling. However, it does not consider the irregularity of tasks. If there is no data dependency between incoming tasks, which means the communication cost is zero, the HEFT algorithm will always assign the highest priority to the task that has the minimum execution time. The MFIT algorithm targets performance in multi-FPGA systems. It always selects the resources with lowest task stack when they have similar suitable level by considering the minimum completion times, time variances and number of allocated tasks. However, the MFIT algorithm is a pure task placement algorithm which solely considers hardware occupancy and architecture, and does not account for the content or properties of each task.

III. Ranking Irregular Tasks

Building on our previous work [9], we provide a generalized procedure for developing a heuristic-based runtime scheduler that can handle resources with arbitrary capacity and allocation representations. The primary goal is to efficiently construct heuristics for irregular multi-dimensional tasks to guide the allocation strategy. An efficient way to construct rankings on modular lattice topologies is by the *height* function [10] which counts interim nodes from the bottom to the current node. *Consistency* is satisfied if the ranking on lattice can label each node to represent a *consistent* distance between that node and the bottom whilst preserving their initial order. The consistence means for any node in the lattice, the length of every path from that node to the bottom will be the same. The difficulty appears when we consider the non-modular lattice structure for general multi-FPGA systems with arbitrary topology: if there is a node which has different interim nodes from itself to bottom by following different paths, then it cannot be ranked by the $height$. We provide a concrete example of this scenario as shown on the left in Fig. 2: Consider the node e - its $height$ value equal to either 3 or 2 based on the different paths from the bottom node a.

To solve this problem we consider a reverse *Birkhoff's representation* [11] on the initial non-modular lattice topology for tasks model on general cluster. A pseudo-code description of this process is shown in **Algorithm 1**. Recall the example shown in Fig. 2, the resulting modular lattice is presented in the right hand side where each node in here has a consist $height$ value. It shows our ranking is *easy computable*, that the processing can be finished within polynomial time O(n^2) by only traversing the initial matrix of orders between tasks. We can assign the ranking value for each task right after generating the corresponding downset. Finally, the height function on downset lattice is *consistent* as ranking on height function is *order-preserving* for the initial relationship between tasks.

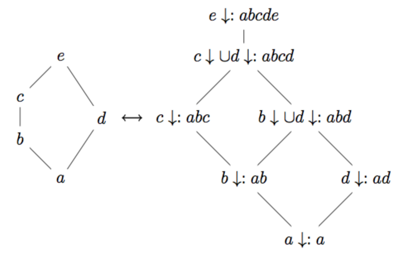

Fig. 2. Lattice representation

Algorithm 1 Birkhoff's representation on Initial task model and ranking

```
1: INPUT: (Set of tasks with initial orders)
2: $N = number\ of\ tasks$
3: for i = 1 to N do
4:     $taskDownset_i$ = {}
5: for i = 1 to N do
6:     for j = 1 to N do
7:         if $taskOrders_j \leq taskOrders_i$ then
8:             $taskDownset_i = taskDownset_i + \{task_j\}$
9:     $taskRank_i = getSize.taskDownset_i$
```

IV. Scheduling Algorithm

Problem Formulation and Abbreviations To consider the underlining problem: we have a certain amount of hardware accelerator resources which can be formulated as a set of $\mathbb{A}$: $Acc_1 ... Acc_M$; To handle a multi-workload scenario S : $W_1 ... W_N$, where each workload W_k contains different number of tasks $\mathbb{T}_k$: $task_1 ... task_{N'}$. The outputs of our algorithm are scheduling solutions derived at runtime which can be formulated as mappings $\mathcal{P}(\mathcal{S})$: $\{\mathbb{T}_k\} \mapsto \mathbb{A}$. Each workload has a completion time: $wTime_k(\mathcal{P}) = max.\ taskTime_i(\mathcal{P})$ Where $taskTime_i$ is the completion time of $task_i$. Based on the different mappings $\mathcal{P}$, the multi-FPGA system will have different total workloads completion time WCT which can be formulated as a higher level function of the mapping $\mathcal{P}$: $WCT(\mathcal{P}) = max.\ wTime_k(\mathcal{P})$ The subject of the algorithm is to give good solutions $\mathcal{P}$ to achieve high system performance represented by reducing WCT. We use N to denote the number of workloads in a multi-workload scenario, N' to denote the number of tasks in a workload and M to denote the number of accelerators. Acc is to denote accelerator. For each task and accelerator, we give it an state (taskState/AccState) which represents whether it has been allocated to run (1) or not (0). Tasks also have a round number (taskRound) which means in which round they are allocated. We use a N'*M matrix (taskSol) to record the solutions of scheduling for each workload, where $taskSol_{i,j} = 1$ means we schedule $task_i$ to run on accelerator Acc_j. ***Ranking-based Scheduling*** The pseudo code of our ranking-based scheduling algorithm is shown in Algorithm 2. To implement the scheduling process, the first step is how to achieve the *fit* mechanism based on rankings from tasks. Our fit mechanism is a variant of the

Algorithm 2 Ranking-based Scheduling Algorithm on multi-FPGA Systems

```
while N > 0 do
  j = 1
  for i = $arg\ max.taskRank_i$ to $arg\ min.taskRank_i$ do
    if $taskState_i$==0 then
      for d = 0 to $M$ do
        if $taskRound_i \geq AccTime_d$ then
          $AccState_d$=0
      count = 0
      while count $\leq taskDim_i$ do
        if $AccState_j$ = 0 then
          $taskSol_{i,j}$ = 1; $AccState_j$ = 1;
          $AccTime_j$ += $taskTime_i$; count += 1
        j += 1
        if j $\geq$ M then
          j $\rightarrow$ the first empty Acc in next device
      $taskState_i = 1$; N = N - 1
      j $\rightarrow$ the first empty Acc in previous device
```

well-known First-Fit-Decreasing (FFD) approach guided by our ranking. As demonstrated by [12], FFD is efficient for resource assignment and we design our fit mechanism based on it. To achieve runtime scheduling, we involve a *Round* mechanism. Another issue is the theoretical upper bound of the performance. A simple theoretical upper-bound here is: $WCT_{upper} \geq max.\{\frac{\sum wTime_k}{M}, max.\{wTime_k\}\}$

V. Software Architecture and Experimental Setup

The runtime scheduling system is responsible for the order of execution and the mapping of tasks to FPGAs. Tasks are all *malleable*, meaning they can be instantiated on any number of compute units in parallel, and any number of tasks may be scheduled concurrently. At runtime, tasks are first ranked by the ranking generator equipped with algorithm **Algorithm 1**. This decision is based on both the static information about the ideal allocation provided from the analysis, and runtime information about the current resource availability, provided by the resource monitor. Finally, the scheduler allocates tasks to FPGAs by algorithm **Algorithm 2** - the round mechanism is triggered periodically based on empirical time slicing.

Hardware Setup Our approach was tested on a testbed with FPGA-based multi-accelerator systems, the MPC-X device produced by Maxeler Technologies [1]. It is comprised of eight DataFlow Engines (DFE) in which each of the DFEs is physically interconnected via a ring topology. All DFEs are same specialized computation resources, each employing a Xilinx Virtex-6 FPGA to support reconfigurable designs and 48GB (or more) RAM for bulk storage. With this setup, applications running on CPU-based machines dispatch computationally intensive tasks to single or multiple DFEs across an Infiniband network. Each DFE cannot be separated to multiple applications simultaneously which means each DFE can only be scheduled to single task in our runtime scheduler. ***Workloads Descriptions*** Reverse Time Migration (RTM) is a practical method commonly used in the Oil and Gas industry to model the bottom of salt bodies in earth's subsurface. Function 11 (F11) is a well-known benchmark [13] to test the performance of Genetic algorithm (GA). A detailed parallel GA kernel design on FPGA can refer to [14]. ***Experimental Strategy*** We run our scheduling architecture and compare the performance with a FIFO approach and our implementation of HEFT and MFIT on both multiple RTM workloads and the parallel-GA benchmark, F11. We evaluate our approach based on three kinds of multi-workload scenarios: *Single-task workloads:* Each workload is made up of a single run of the benchmark, which consists of a number of parallel elements of varying irregularity; *Multi-task workloads:* Each workload is made up of a random number of runs of the benchmark between 2 and 20, operating on different data. Each run consists a number of parallel elements of varying irregularity; *High irregularity workloads:* Further, we evaluate a multi-task scenario where the tasks considered are highly irregular in size. ***Experimental Metrics*** *Workload Completion Time (WCT)* is the performance metric applied for each multi-workload scenario in our experiment. WCT is the makespan of the system - As for RTM workloads, it represents the workloads completion time for all tasks in all workloads submitted; As for F11 benchmark, it represents the final time when all optimal solutions are found for different domain of input sizes. *Percentage of Upper-Bound* shows the percentage between the solution of our tested approaches and the theoretical performance upper-bound which is formulated in above section. The irregularity of tasks presented the difference of suitable configurations - the number of accelerators and their topology needed. It is hard to define the underlining irregularity from attributes of data directly as it depends on the workloads composition and kernels which is used to execute it. Instead, we give an empirical metric of irregularity in workloads to demonstrate our results. $Irg(W)$ to denote the irregularity level of each workload W_k as follows: $Irg(W_k) = \sum_{i=1}^{N'}(\frac{taskTime_i}{\sum taskTime_i}) * (\frac{taskTime_i - \overline{taskTime}}{\overline{taskTime}})^2$ Where $\sum taskTime_i$ denotes the total execution time of tasks in workload W_k whilst $\overline{taskTime}$ means the average execution time of tasks.

VI. Results

Results for Multi-task Cases The experimental results of RTM workloads and GA F11 benchmark on different workload scenarios through different irregularity levels are shown in the Figure 3. The $Ranking$ approach consistently outperforms all other approaches in multi-task scenarios across levels of irregularity. For RTM workloads, $Ranking$ maintains 99.7% to 99.8% of upper-bound of performance across levels of irregularity. This represents a significant improvement of between 5% and 7% over the commercially implemented FIFO scheme. Our implementation of the HEFT approach for this task performs fairly well here, but only reaches 97.5% to 98% of the maximum possible performance. MFIT performs similarly

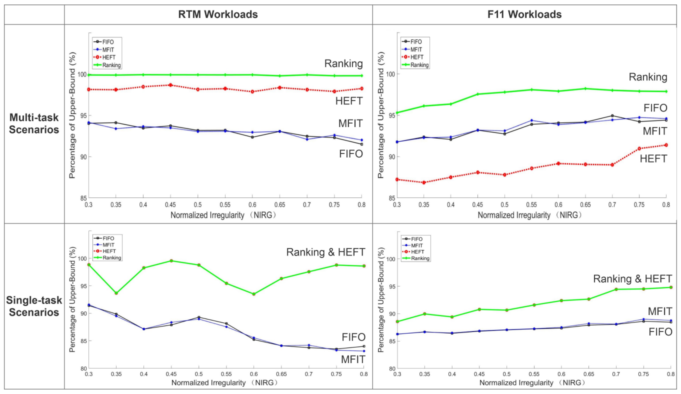

Fig. 3. Experimental results on different workload scenarios

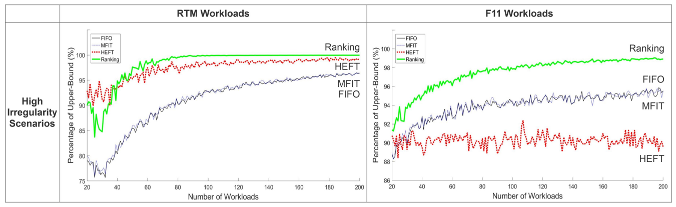

Fig. 4. Experimental results on high irregularity scenarios

to FIFO. For F11 workloads, with normalized irregularity around 0.3, *Ranking* already reaches more than 95% of the upper bound, while both FIFO and MFIT are around 92% and HEFT 87%. As irregularity increases to 0.55, the *Ranking* approach obtains 98% and does not decrease. Both FIFO and MFIT reach a maximum of 93.5%, while in this case, HEFT only obtains between 86% and 88% of the upper bound. Over both scenarios, the *Ranking* approach provides the best performance. ***Results for varying workload numbers in High Irregularity Scenarios*** Figure 4 shows the results where only highly irregular workloads are considered, and where the number of workloads simultaneously present for computation is varied. In general, scheduling becomes easier as the number of workloads increases as there are more tasks of varying size which allow holes in the schedule to be more easily filled. This intuition is borne out across all considered approaches. *Ranking* generally performs better than all other approaches across both RTM and F11 workloads. For RTM workloads, at a low number of workloads, HEFT performs a little better than ranking. For example, when the number of workload is around 30, HEFT can reach around 93% whilst *Ranking* only get around 85%. This is due to the HEFT algorithm preferring longer running tasks running on a small number of accelerators, rather than ranking, which prefers shorter running tasks running on a larger number of accelerators. The overhead of changing tasks more quickly in low workload size scenarios leads to lower performance with the ranking approach. Even in this case, *Ranking* is still outperforms MFIT and FIFO, both of which obtain about 76% of the available performance. *Ranking* consistently outperforms HEFT as the number of workloads increases past 50,quickly achieves 99% or available performance. For F11 workloads, *Ranking* outperforms all other three approaches across all numbers of workload. The results vary from 91% to more than 98% of the upper bound of performance. MFIT and FIFO achieve around 93% when the number of workloads is greater than 60. HEFT fluctuates around 90% across differing numbers of workload.

VII. Conclusion

We show our ranking based dynamic scheduler is able to outperform the FIFO scheduler used in the Maxeler Orchestrator [1] and the the most relevant research schedulers, HEFT and MFIT in multi-task scenarios. The approach was tested on both RTM and parallel GA benchmark workloads. The performance gains between our approach against other approaches are more significant for high irregularity and multi-task scenarios on both workloads. In scenarios where most workloads contain similar sizes of task, our approach maintains the same or better performance. In these simpler cases, the existing approaches are already perform well. We present a novel dynamic scheduling approach for multi-FPGA systems, based on a partial order representation of tasks and a ranking methodology. We show how our dynamic scheduling approach, which models and ranks irregular workloads, outperforms existing approaches in multi-task scenarios, while retaining the same performance in single-task scenarios. This work shows the prevalence of FIFO schedulers in industrial and research accelerator systems is not without cause – FIFO performs well in a mostly homogeneous single-task environment. However, in the presence of irregular tasks and FPGA-based multi-accelerator resources, a better method is required to retain high levels of performance.

References

[1] Maxeler, *https://www.maxeler.com*, 2016.
[2] C. Jing *et al.*, "Energy-efficient scheduling on multi-fpga reconfigurable systems," *Microprocessors and Microsystems*, vol. 37, no. 6, pp. 590–600, 2013.
[3] H. Topcuoglu *et al.*, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE TPDS*, vol. 13, no. 3, pp. 260–274, 2002.
[4] HARNESS, "The harness platform: A hardware- and network-enhanced software system for cloud computing," tech. rep., Nov. 2015.
[5] CloudLightning, "Cloudlightning position paper," 2016.
[6] C. Margiolas and M. F. O'Boyle, "Palmos: A transparent, multi-tasking acceleration layer for parallel heterogeneous systems," in *ICS*, pp. 307–318, ACM, 2015.
[7] T. Hussain *et al.*, "Amc: Advanced multi-accelerator controller," *Parallel Computing*, vol. 41, pp. 14–30, 2015.
[8] T. Hussain *et al.*, "Ppmc: Hardware scheduling and memory management support for multi accelerators," in *FPL*, pp. 571–574, IEEE, 2012.
[9] T. Yu *et al.*, "Relation-oriented resource allocation for multi-accelerator systems," in *ASAP*, pp. 243–244, IEEE, 2016.
[10] M. P. Schellekens, "The correspondence between partial metrics and semivaluations," *TCS*, vol. 315, no. 1, pp. 135–149, 2004.
[11] G. Birkhoff, *Lattice theory*, vol. 25. American Mathematical Soc., 1940.
[12] R. Panigrahy *et al.*, "Heuristics for vector bin packing," *research. microsoft. com*, 2011.
[13] D. A. Coley, *An introduction to genetic algorithms for scientists and engineers*. World scientific, 1999.
[14] L. Guo *et al.*, "Parallel genetic algorithms on multiple fpgas," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 4, pp. 86–93, 2016.