TF-PSST : A Spatio-Temporal Scheduling Approach for Multi-FPGA Parallel Heterogeneous Architecture in High Performance Computing

Aditya Das, Rishab Bhattacharyya and Pramit Ghosh

Abstract—This work is a proposed architectural prototype in the field of High Performance Computing (HPC). Intel Altera DE4 and Altera DE5a - Net FPGA boards were used as functional processors in our designed system. We further explore Peripheral Component Interconnect (PCI) Express communication and amalgamate the transfer of data through PCIe to two different kinds of FPGAs at the same time using a proposed scheduling algorithm called TF-PSST : Time First Power Second Scheduling Technique. This significantly improves efficiency of the system by reducing execution time and because of the heterogeneous nature of the architectural prototype, we also found a way to increase the hardware resource utilisation.

Index Terms—High Performance Computing, FPGA, Heterogeneous Computing, Peripheral Component Interconnect.

1 Introduction

The need to increase the computational speed of processing tasks along with saving power has become the need of the hour in the field of Heterogeneous Systems and High Performance Computing (HPC). In the last few years, interest in High Performance Computing has increased substantially across industries and academia [1], [2]. The emphasis on concurrent processing has also increased the acceptability of heterogeneous environments for developing HPCs [3]. The use of Field Programmable Gate Arrays (FPGAs) for achieving high performance computing can deliver enormous results [4]. Spatio-Temporal Scheduling Algorithms aims at representing information both spatially(space) and temporally(time) [5]. This can be used to an advantage especially when spatio-temporal scheduling have been known to increase efficiency and provide optimized solutions when it comes to computational speed, energy efficiency or data storage [6]. In our work, we implemented an architectural prototype that was executed in the host PC, for a user to enter data for a particular task of his choice from a predefined list of tasks. In a multi-FPGA system, in order to improve the resource utilization of the system as much as possible, it is necessary to reasonably allocate the available hardware resources and balance the utilization rate of fine-grained hardware resources in the FPGA. The input instruction and data of the selected task would be sent to any of the FPGAs as specific processing elements in a spatio-temporal approach. A newly proposed scheduling approach (TF-PSST) has been used in order to transfer the given data and tasks into any connected processing element. Evaluation of the performance of our implemented prototype has shown significant decrease in processing time and increased hardware resource utilisation of the system. The overall contribution of the paper can be summed up in the following way. Firstly, we have defined a hypothesis for the scheduling and placements of tasks in a multi-FPGA system. Secondly, we propose a scheduling approach to improve resource utilisation by reducing execution time, decreasing power consumption, improving hardware utilisation and decreasing the latency. And finally, we extensively study the impacts of our proposed scheduling approach.The main novelty of this work lies in the domain of high performance computing wherein we used a completely new approach of allocating tasks into different kinds of FPGAs using spatio-temporal scheduling approach through a PCIe bus.

- .
A.Das is from L'École nationale supérieure des mines de Saint-Étienne, France. R. Bhattacharyya is from RWTH Aachen, Germany. P. Ghosh is from RCC Institute of Information Technology, India. Email(s): aditya.das@etu.emse.fr, rishab.bhattacharyya@rwth-aachen.de, pramitghosh2002@yahoo.co.in
-

Manuscript received April 19, 2005; revised August 26, 2015.

2 Related Works

The authors in [7] implemented two algorithms from the Rodinia benchmark to analyse power and speed of the overall system. The authors claim that their work showed a speed increase of about 5.3x. However, their overall system had a reduction of just 21 W of power which is not a significant reduction when we take energy efficiency of the system into account. Similarly, the increase in speed was also not significant. There has of course been various traditional algorithms designed for task scheduling, such as Min-Min [8], Max-Min [9] and first-in-first-out (FIFO) [10]. While Min-Min and Max-Min fail to utilise the available resources properly thereby causing load imbalances, FIFO performs the tasks in the order of their submissions. This of course has several of its own drawbacks with regards to proper utilisation of time as a resource. If a submitted

task would be taking a long time to be performed, the tasks submitted after that would all be on hold, thereby potentially paralysing the system. Exhaustive-search-based task-to-FPGA mapping is nearly impossible. This is because the complexity of the algorithm would grow exponentially with the increase of tasks. Therefore, determining the type of scheduling approach to be used for our work is important to establish an efficient system.

3 Problem Description

3.1 System Model

The system model provides us with an illustrative idea behind the working mechanism of the proposed architectural prototype. There are multiple FPGAs connected to the host PC through a 16 lane PCIe.

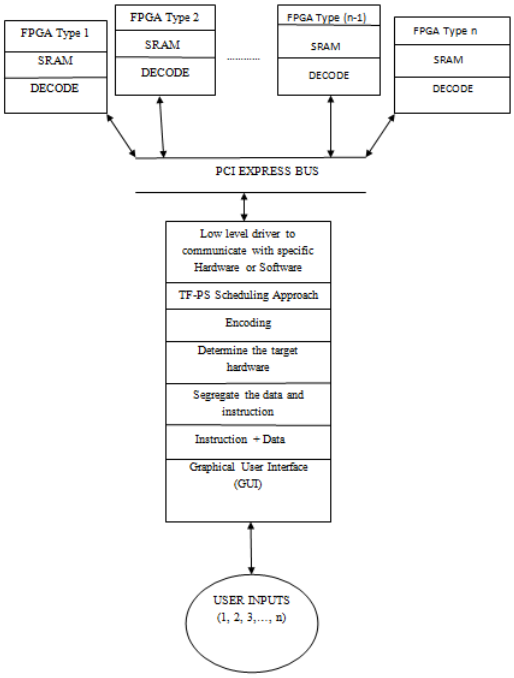

Fig. 1: The basic System Model of our designed architecture

The host server employs a Graphical User Interface (GUI) in order to interact with the user. The prototype here works in six specific layers. Firstly, the user interacts with the system through a designed GUI. The user selects the tasks of his choice from a predefined task list and also enters the data into the system which goes through the second layer, where the supplied data and instructions are recognised, and into the third layer wherein segregation of instructions and the data takes place. The fourth layer is where specific encoding of the data and instructions takes place in accordance with PCIe protocols. The fifth layer is where our spatio-temporal scheduling algorithm decides on which hardware the task will be processed in for the greatest efficiency, based on its computation time and power consumption. The objective of TF-PS scheduling technique is to maximize hardware resource utilisation with minimum execution time. In the sixth and final layer, the specific hardware is targeted to execute the instructions for the supplied data.

3.2 Scheduling Problem Model

Here, TA_c is defined as the set of c tasks defined in the system. Each of these tasks can be submitted by the user simultaneously. When a task is executed, it must first be configured in the logical resources available in the FPGAs. This process leads to a certain reconfiguration overhead. However, if another user executes the same task and it is executed in the same reconfigurable region, there would be no reconfigurable overhead in the system [11]. Hence, our objective primarily is to minimise the task makespan of the system by designing an efficient algorithm for task-to-FPGA placement, while taking into consideration, the time, hardware resource constraints, overhead configuration time and power consumption.

4 Design Implementation

In the protoptype implementation of the System Model, we used Intel Altera DE4 and Intel Altera DE5a-Net as two FPGA models which were directly connected to the ”host PC’s” motherboard through a PCIe bus. PCIe is a serial point-to-point connection, it is faster than its parallel predecessor and currently offers the speed of 34 GB/seconds [12]. The use of the newly proposed TF-PSS approach also further increased hardware resource utilisation of the overall system, apart from a significant speed-up. The use of two different kinds of FPGAs from two different families as two parallel processors increases the heterogeneity of the system.

4.1 TFPS Scheduling Approach: An Example

In this scheduling strategy we schedule the task by prioritizing *Time* first and *Power* second. Let us assume, User 1 enters multiple Tasks through the GUI. We will then compare the execution time of Task 1 in Device 1(First Processing element) with the execution time of Task 1 in the Device 2 (Next processing element). If the waiting queue(P) does not contain the device where the execution time is less, we will allocate the task to that particular device and place the device in the waiting queue. If the waiting queue already contains the Device where the execution time is less we will allocate the task to the next best processing element where the execution time of the task is the least and subsequently place the device in the waiting queue.If the execution time of a task is same in two processing elements, we will do same comparison on the basis of power consumed. If there are no available processing elements left for a particular task to be mapped into, we store that task in a task queue. Then we check if the summation of the wait time and execution time is less than the deadline or not . If less than the deadline we map the task to processing element where the least time is consumed, if the processing element is free, else we map it to the next best processing element where time is optimized. If the summation of the wait time and execution time is greater than the deadline we scale the frequency of the best available processing element by 2X.

5 Experiment and Result

5.1 Execution Time Comparison

ISCAS Benchmarks 85 (C6288 and C7552) and the Nearest Neighbour Algorithm and Lava MD from the Rodinia

Algorithm 1: TFPS Scheduling Strategy for Processing Element mapping

Input: Users U_a, D_b Devices, TA_c Tasks, P_o Power of Devices, T is Time, Task Queue T_q, Time Sorted T_d, Device Sorted D_d, Waiting Queue P
Output: Task to Processing Element (PE) mapping

1 Initialize Waiting Queue and Task Queue = ;
2 **for** *(each a in* $Users$*)* **do**
3 **for** *(each b in* $Devices$*)* **do**
4 **for** *(each c in* $Tasks$*)* **do**
5 **for** *(each d in Time Sorted)* **do**
6 **if** *(Time of* $Devices(D_b, c, a)$ *< Time of* $Devices(D_{b+1}, c, a)$*)* **then**
7 **if** *(Waiting Queue(P)* $!= D_b$*)* **then**
8 Map the task to D_b;
9 **else if** *(Waiting Queue(P)* $!=$ *Devices(*D_d*))* **then**
10 Map the task to D_d;
11 Waiting Queue $= D_d$;
12 **else**
13 Task Queue = Task(TA_c);
14 **if** *(Wait Time + Execution Time < Deadline)* **then**
15 **if** *(Waiting Queue(P)* $!=$ *Device(*D_b*))* **then**
16 Map the task to D_b;
17 **else**
18 Map the task to D_d;
19 **end**
20 **else**
21 Scale the frequency of the best available device by $2X$;
22 **end**
23 **end**
24 **end**
25 **if** *(Time of* $Devices(D_b, c, a)$ *> Time of* $Devices(D_{b+1}, c, a)$*)* **then**
26 **if** *(Waiting Queue(P)* $!= D_{b+1}$*)* **then**
27 Map the task to D_{b+1};
28 **else if** *(Waiting Queue(P)* $!=$ *Devices(*D_d*))* **then**
29 Map the task to D_d;
30 Waiting Queue $= D_d$;
31 **else**
32 Task Queue = Task(TA_c);
33 **if** *(Wait Time + Execution Time < Deadline)* **then**
34 **if** *(Waiting Queue(P)* $!=$ *Device(*D_{b+1}*))* **then**
35 Map the task to D_{b+1};
36 **else**
37 Map the task to D_d;
38 **end**
39 **else**
40 Scale the frequency of the best available device by $2X$;
41 **end**
42 **end**
43 **if** *(Time of* $Devices(D_b, c, a)$ *= Time of* $Devices(D_{b+1}, c, a$ *and Power of* $Devices(D_b, c, a)$ *<* *Power of* $Devices(D_{b+1}, c, a)$ *)* **then**
44 Repeat steps 8 to 22;
45 **end**
46 Repeat steps 26 to 42;
47 **end**
48 **end**

Benchmarks, along with a document classification algorithm were implemented in the prototype.

Fig 2 shows the graphical representation of execution time (which was measured using Quartus Time Analyser) between [8] and our prototype for Nearest Neighbour (NN), LAVA MD (LM) and Document Classification (DC). The total execution time for each task for our prototype was measured 10 times and the average of the results were taken into account for comparison. The graphs clearly note the increase in performance by reducing execution time for the task at hand significantly. We implemented ISCAS Benchmarks 85, C6288 and C7552 in our prototype and compared the execution times with respect to the execution time of a normal PCIe-FPGA architecture, without the scheduling algorithm. Our results show that all the tasks had a speed-up of 2.03X and 1.96X respectively. For Fast Fourier Transform (FFT), we had a speed-up of 1.08X.

TABLE 1: Comparison of Execution Time of different tasks between [8] and the proposed FPGA-based PCIe (using TF-PSS)

Task	Model Device	Execution Time(in seconds)	% -	Speed-up
N.N.	Model Used in [8]	0.000444	-	
N.N.	Our Prototype	0.000139	68%	3.19X
L.M.	Model Used in [8]	201.447555	-	
L.M.	Our Prototype	168.9014566	16%	1.19X
D.C.	Model Used in [8]	7.273355	-	
D.C.	Our Prototype	6.8772555	5%	1.05X
c6288	Model used in [8]	-	-	
c6288	Our Prototype	1.2673	-	
c7552	Model used in [8]	-	-	
c7552	Our Prototype	1.5369	-	
FFT	Model in [8]	-	-	
FFT	Our Prototype	3.6350	-	

5.2 Hardware Utilisation

Hardware Utilisation of the system is reported in the following table. As it is noted, the highest utilisation occurs in LAVA MD (L.M.), while the lowest overall utilisation of hardware resources is noted in Fast Fourier Transform (FFT). All the algorithms were manually optimised for overall better performance of the system.

TABLE 2: Hardware Resource Utilisation of the system

	D.C.	L.M.	N.N	c6288	c7552	F.F.T.
Logic Utilisation	73%	83%	89%	77%	71%	69%
Dedicated Logic Reg.	36%	41%	39%	33%	31%	28%
Memory Blocks	79%	74%	66%	68%	70%	61%
DSP Blocks	16%	67%	61%	67%	61%	66%

5.3 Comparison with non-parallel Heterogeneous non-High Performance Computing

If the FPGAs mentioned in the system model are connected in a serial way, the scheduling approach as well the result of the same would be significantly different than the one presented above. In this case, only FPGA A would be connected to the host PC through the PCI Express. FPGA B would be connected to FPGA A and thereon until FPGA (n-1). In this scenario, the FPGA would have to be reconfigured after each time it receives the instructions to perform a task, would of course increase the reconfigurable overhead. We performed the same experiment by comparing a couple of tasks being scheduled in a non-parallel way to our original model. The algorithmic tasks implemented were once again the Nearest Neighbour (N.N.), LAVA MD (L.M.), Document Classification (D.C.), Fast Fourier Transfer (F.F.T.) and ISCAS benchmarks c6288 and c7552. Table 3 shows the result we received through this comparison.

5.4 Proof of Algorithmic Optimality of TF-PSST

The algorithm is always supposed to map a task (T) to a device (D) based on the shortest execution time and lowest power consumption. We prove the algorithm is optimal through contradiction. Let us assume that the task **N** has the

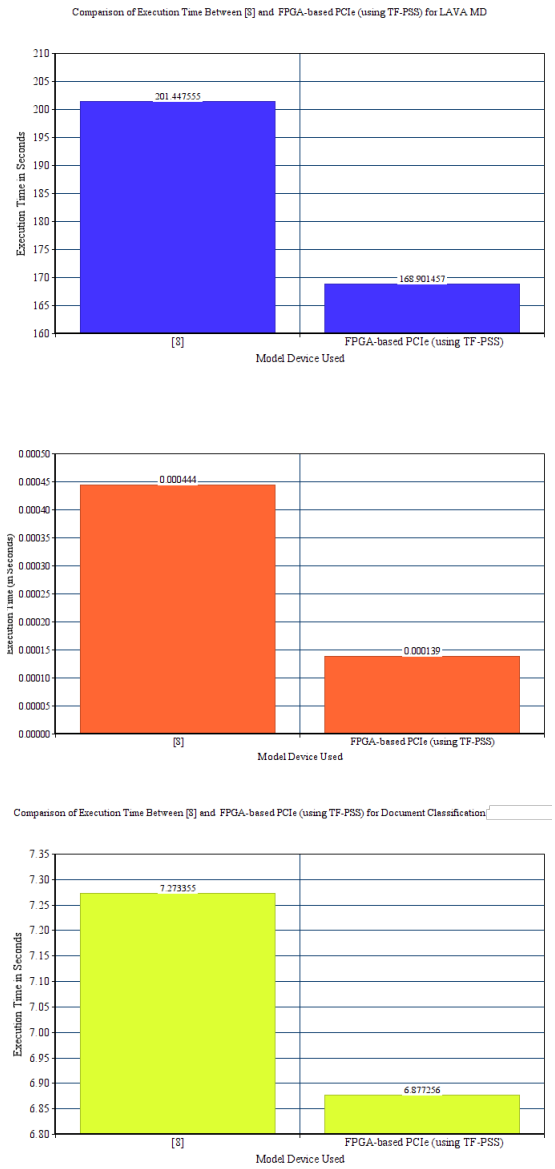

Fig. 2: A graphical representation of execution time comparison of the benchmarks implemented

TABLE 3: Comparison of Execution Time of different tasks between Non-Parallel Architectural Scheduling (NAS) and the proposed Multi-FPGA-based PCIe (using TF-PSS)

Task	Device Model	Execution Time(in seconds)	% +
N.N.	NAS	0.002363	1600 %
N.N.	Our Prototype	0.000139	
L.M.	NAS	2195.713	1199.99 %
L.M.	Our Prototype	168.9014566	
D.C.	NAS	130.6678	1799.882%
D.C.	Our Prototype	6.8772555	
c6288	NAS	51.95	3999.266%
c6288	Our Prototype	1.2673	
c7552	NAS	50.717	3199.95%
c7552	Our Prototype	1.5369	
FFT	NAS	123.59	3300%
FFT	Our Prototype	3.6350	

shortest execution time and the lowest power consumption in device **D**. Consider a solution **S** where **N** is not mapped to **D**. Replace **q**, where **q** being mapped to **D** takes the longest execution time. This gives us a new solution **S′**. Now **T(N)** $<$ **T(q)** (where T(N) is the execution time for N and T(q) is the execution time for q), because **T(q)** is always more than **T(x)**, where **x** is a task mapped to **D** at the head of the waiting queue (**P**). Also, **T(N)** $<$ **T(x)**, so we have the follows:

$$\sum S' < \sum S \tag{1}$$

Hence, we have a contradiction. We can conclude from the above proof that the optimal solution must always map **N** to **D**.

6 Conclusion and Future Work

Our findings show that our proposed model of architecture for HPC significantly improves the system's performance and also reduces the overall power consumption.This is the first algorithmic, architectural prototype of its kind using fully reconfigurable FPGA mode. We wish to further decrease the latency and the execution time of the model by studying the implications under partially reconfigurable FPGA mode.

References

[1] (2019) Hpcwire. [Online]. Available: http://www.hpcwire.com
[2] (2019) insidehpc. [Online]. Available: https://insidehpc.com
[3] E. Ilavarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments," *Journal of Computer sciences*, vol. 3, no. 2, pp. 94–103, 2007.
[4] M. C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with fpga-based computing," *Computer*, vol. 40, no. 3, pp. 50–57, 2007.
[5] H. Yuan, J. Bi, and M. Zhou, "Spatio-temporal scheduling of heterogeneous delay-constrained tasks in geo-distributed green clouds," in *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*. IEEE, 2019, pp. 287–292.
[6] Z. W. Bhatti, N. R. Miniskar, D. Preuveneers, R. Wuyts, Y. Berbers, and F. Catthoor, "Memory and communication driven spatio-temporal scheduling on mpsocs," in *2012 25th Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, 2012, pp. 1–6.
[7] O. Segal, N. Nasiri, M. Margala, and W. Vanderbauwhede, "High level programming of fpgas for hpc and data centric applications," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2014, pp. 1–3.
[8] J. Li, M. Qiu, J. Niu, W. Gao, Z. Zong, and X. Qin, "Feedback dynamic algorithms for preemptable job scheduling in cloud systems," in *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, vol. 1. IEEE, 2010, pp. 561–564.
[9] S. Devipriya and C. Ramesh, "Improved max-min heuristic model for task scheduling in cloud," in *2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*. IEEE, 2013, pp. 883–888.
[10] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima, "Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2004, pp. 210–232.
[11] R. Cattaneo, R. Bellini, G. Durelli, C. Pilato, M. D. Santambrogio, and D. Sciuto, "Para-sched: A reconfiguration-aware scheduler for reconfigurable architectures," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 2014, pp. 243–250.
[12] I. Corporations. (2019) Intel® pci and pci express*. [Online]. Available: https://www.intel.ie/content/dam/doc/case-study/intel-pci-pci-express-3-case-study.pdf