A Dynamic Load Balancer for a Cluster of FPGA SoCs

Farnam Khalili*† and Roberto Giorgi*

* *Department of Information Engineering and Mathematics, University of Siena, Italy*
† *Department of Information Engineering, University of Florence, Italy*

ABSTRACT

In many-core systems to achieve maximum performance, it is desirable to produce many tasks more than the cores and efficiently distribute those tasks among available resources. Software load balancers will provide enough performance as long as the number of jobs is big enough in comparison with the load balancing overhead. To mitigate this overhead, delegating load balancing to an accelerator will improve the performance of such architectures. This paper presents a hardware dynamic load balancer module implemented on the FPGA Zynq Ultrascale+ and is based on the semi-work-stealing[2] scheduling. The load balancer is specifically designed for Data-Flow-Threads (DF-Threads) and can support multi-core and multi-node computing architectures. The performance of the design is initially examined through a simple "stress-test" that generates threads (the Recursive-Fibonacci program) on a two-nodes FPGA cluster.

KEYWORDS: FPGA, Zynq Ultrascale, Distributed Shared Memory, Load Balancing, Dataflow.

1 Introduction

The efficient distribution and scheduling of tasks in a multi-thread distributed system is yet a compromising effort and open research topic. A well-known and practical method of this kind of parallel computation is "work-stealing" [CL05, ABB00], in which processors requiring jobs steal computational tasks from other processors. In the full software version of such schedulers, if the granularity of the problem is not big enough, the overhead arose from the load balancer and the communication is comparable and reduces the efficiency of the processing power. In this paper, we present a dynamic load balancing engine fully implemented on the Programmable Logic (PL), of a Xilinx Zynq Ultrascale+ board (AXIOM board) [GKP19a, GKP18]. This module performs dynamic load balancing based on a semi-work-stealing method, for a Data-Flow Threads (DF-Threads) [Gio18] distributed system partially implemented on the PL [PKG18]. Many applications can expose their thread-level parallelism to the DF-thread scheduler to efficiently harness maximum available computing

[1]E-mail: {khalili,giorgi}@dii.unisi.it
[2]despite the work-stealing, in which the job requester doesn't ask for the permission, in semi-work-stealing, the job provider should allow the job requester before migrating the available jobs

resources [VSG19, SVG19].

In order to stress the capability of managing many threads of our proposed load balancer, we use a Recursive-Fibonacci benchmark on the two nodes (boards). All the modules of the hardware design are implemented concerning our methodology [GKP19c] via the High-Level Synthesis (HLS) tool. Zynq Ultrascale+ SoC comprises a Processing System (PS), and a Programmable Logic (PL) part. To interface between PS and the PL, we evaluated the performance of the interface [KG19], and decided to choose a register-based AXI-lite interface (to control and monitor the hardware IPs) and a FIFO-based AXI4 interface (to pass high throughput data) [KPG18].

2 Background (DF-Threads)

DF-Threads are fine-grained parallel threads based on the DataFlow paradigm that can be distributed among a cluster of multi-core systems [GF14]. In this implementation, the computation threads are isolated from the job scheduler by offloading the communication patterns and load balancing, which eases significantly processors burden. Each DF-Threads executes in sequential by the CPU (here ARM). Certain instructions to be executed are associated to the memory regions so-called DF-Frames, which their executions initiate once all of its inputs are available. The inputs can be provided by other threads executing locally or remotely. Instruction pointers of those frames that are ready to be executed (FPs) are stored in a circular buffer called Ready Frame Queue (RFQ). A design space exploration tool-set to analyze the key metrics and eventually evaluate this model is presented in [GKP19b].

3 Dynamic Load Balancer

Figure 1 shows our novel hardware DF-Thread scheduler implemented on the AXIOM board. Local Scheduler comprises several Finite State Machines (FSM), to realize the novel DF-threads API functionality. Load Balancing engine is responsible to distribute the ready Frame Pointers (FPs), to available local cores or remote nodes. The load balancing algorithm is similar to the work-stealing with this difference that here, the node that asks for the jobs will get the available job until the overloaded node permits for the job migration. In this model, each job is one FP.

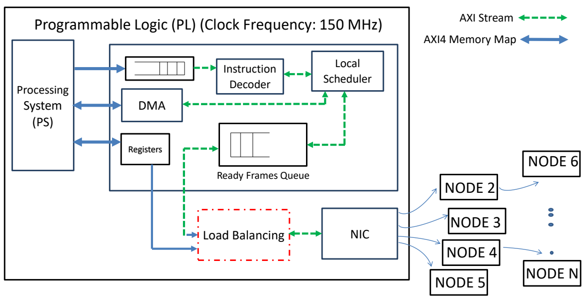


Figure 1: A simplified block diagram of our proposed scalable and reconfigurable DF-Threads Scheduler Co-Processor implemented on the AXIOM board (with four CPUs). The focus of this paper is the Load Balancing module which is shown with a dotted rectangle.

Those nodes in which their RFQ is "Almost Empty" send a "Job Request" message to their neighbors. This message will be propagated through the cluster until one node has an RFQ with the "Almost Full" flag enabled. This node sends an ACK message to the job requester, then the job requester will accept the available jobs by an ACCEPTED message (Figure 2). Finally, the job provider will perform an RDMA (Remote Direct Memory Access) [GMV+16] to migrate a certain number of FPs to the job requester. The number of migrated FPs, and RFQ thresholds (Almost Empty, Almost Full) are set through a register dedicated to this module.

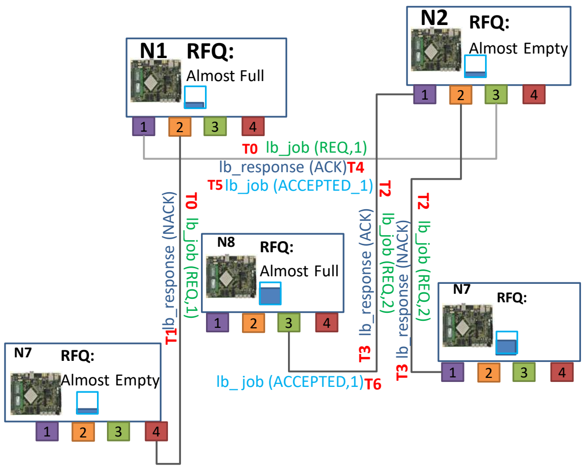
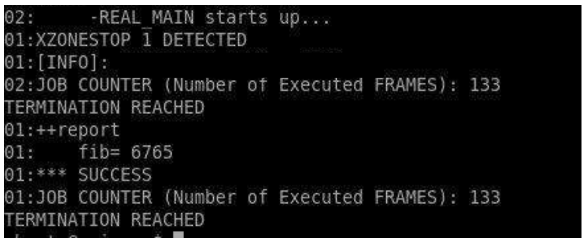


Figure 2: Load balancing message passing across the nodes

4 Test

To initially test the design through a simple “stress-test”, we have implemented a Recursive-Fibonacci benchmark that generates threads on a two-nodes FPGA cluster, with respect to our DF-threads APIs [GS15]. The input size of the Recursive-Fibonacci is 20, which generates 266 number of threads in total. As can be seen in Figure 3, the number of threads are properly distributed among the cluster nodes. Thus, this verifies the correct functionality of our proposed dynamic load balancer.



```
02:     -REAL_MAIN starts up...
01:XZONESTOP 1 DETECTED
01:[INFO]:
02:JOB COUNTER (Number of Executed FRAMES): 133
TERMINATION REACHED
01:++report
01:     fib= 6765
01:*** SUCCESS
01:JOB COUNTER (Number of Executed FRAMES): 133
TERMINATION REACHED
```

Figure 3: Running Recursive-Fibonacci with the input size of 20, as an stress-test that generates totally 266 threads. The proposed hardware load balancer distributes efficiently these threads among a two-nodes FPGA cluster.

5 Conclusion

In this study, we illustrated a dynamic load balancer fully implemented on the PL part of a Zynq Ultrascale+ FPGA (AXIOM board). Dynamic load balancer deploys messages passing through the RDMA messages across the cluster, that decouples the processor from an extra burden of job scheduling.

References

[ABB00] Umut A. Acar, Guy E. Blelloch, and Robert D. Blumofe. The data locality of work stealing. In *Proceedings of the Twelfth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–12, New York, NY, USA, 2000. Association for Computing Machinery.

[CL05] David Chase and Yossi Lev. Dynamic circular work-stealing deque. In *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 21–28, New York, NY, USA, 2005. Association for Computing Machinery.

[GF14] R. Giorgi and P. Faraboschi. An introduction to DF-Threads and their execution model. In *IEEE MPP*, pages 60–65, Paris, France, Oct. 2014.

[Gio18] R. Giorgi. Scalable embedded computing through reconfigurable hardware: comparing df-threads, cilk, OpenMPI and jump. *ELSEVIER Microprocessors and Microsystems*, 63, Aug. 2018.

[GKP18] R. Giorgi, F. Khalili, and M. Procaccini. Energy efficiency exploration on the zynq ultrascale+. In *IEEE Proc. 30th Int.l Conf. on Microelectronics (ICM)*, pages 52–55, Sousse, Tunisia, Dec. 2018.

[GKP19a] R. Giorgi, F. Khalili, and M. Procaccini. Axiom: A scalable, efficient and reconfigurable embedded platform. In *IEEE Proc. Design, Automation and Test in Europe (DATE)*, pages 1–6, Florence, Italy, Mar. 2019.

[GKP19b] R. Giorgi, F. Khalili, and M. Procaccini. A design space exploration tool set for future 1k-core high-performance computers. In *ACM RAPIDO Workshop*, pages 1–6, 2019.

[GKP19c] R. Giorgi, Farnam. Khalili, and Marco Procaccini. Translating timing into an architecture: The synergy of cotson and hls (domain expertise – designing a computer architecture via hls). *Hindawi - International Journal of Reconfigurable Computing*, Dec. 2019.

[GMV$^+$16] R. Giorgi, S. Mazumdar, S. Viola, P. Gai, S. Garzarella, B. Morelli, D. Pnevmatikatos, D. Theodoropoulos, C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. Jimenez-Gonzalez, and X. Martorell. Modeling multi-board communication in the axiom cyber-physical system. *Ada User Journal*, 37(4):228–235, December 2016.

[GS15] R. Giorgi and A. Scionti. A scalable thread scheduling co-processor based on data-flow principles. *ELSEVIER Future Generation Computer Systems*, 53:100–108, Dec. 2015.

[KG19] F. Khalili and R. Giorgi. A soft-ip for performance measuring of the zynq ultrascale+ cpu/fpga interface. In *HiPEAC ACACES-2019*, pages 5–8, Fiuggi, Italy, July 2019. poster.

[KPG18] F. Khalili, M. Procaccini, and R. Giorgi. Reconfigurable logic interface architecture for cpu-fpga accelerators. In *HiPEAC ACACES-2018*, pages 1–4, Fiuggi, Italy, July 2018. poster.

[PKG18] M. Procaccini, F. Khalili, and R. Giorgi. An fpga-based scalable hardware scheduler for data-flow models. In *HiPEAC ACACES-2018*, pages 1–4, Fiuggi, Italy, July 2018. poster.

[SVG19] Amin Sahebi, Lorenzo Verdoscia, and Roberto Giorgi. A data-flow approach to accelerate real-valued fast fourier transform. In *HiPEAC ACACES-2019*, pages 155–158, Fiuggi, Italy, July 2019. poster.

[VSG19] L. Verdoscia, A. Sahebi, and R. Giorgi. A data-flow methodology for accelerating fft. In *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2019.