A Data-Flow Execution Engine for Scalable Embedded Computing

Marco Procaccini, Roberto Giorgi

Department of Information Engineering and Mathematics, University of Siena, Italy[1]

ABSTRACT

Nowadays embedded systems are increasingly used in the world of distributed computing to provide more computational power without having to change the whole system and the programming model. We propose a DataFlow Execution Engine (DEE) to spawn asynchronous, data-driven threads, among embedded cores to achieve a seamless distribution of threads without the need of using a distributed programming model. Our idea relies on the creation of a hardware scheduler that can handle creation, thread-dependency, and locality of many fine-grained tasks. We present an initial evaluation of our DEE that is suited for FPGA implementation. Our initial results show the importance of a hardware based support for such thread execution model.

KEYWORDS: DataFlow; embedded computing; hardware scheduler; multi-thread;

1. Introduction

Thanks to the continued evolution of the processor design, more and more cores are combined in a chip and when the single chip performance is not any more sufficient, a distributed architecture is a potentially interesting solution. One of the major open problems of distributed systems nowadays is to find the right balance between number of threads, runtime overhead and scalability. Models like DataFlow [3] [1] allow us to create many asynchronous threads that can be run in parallel. These threads may encapsulate the data to be processed, their dependencies and, once completed, write their output for other threads. Data-Flow Threads (DF-Threads) [9] can provide a complete decoupling between execution and memory accesses [1]. A hardware "engine", which can manage an appropriate distribution of the threads across the cores of a distributed system, may allow keeping a shared-memory programming model while distributing the computation. This can provide better scalability.

According to Tan [2], a software scheduler cannot handle the fine-grain task scheduling and dependency in a fast and scalable way, causing a fast downgrade of the speedup. A hardware scheduler that handles many tasks quickly and efficiently appears to be a better solution.

Our work aims to use embedded boards based on multi-core SoCs, including memory FPGA, to build a distribute system based on a Data-Flow execution model managed by a hardware scheduler. In the first part of our work (presented here), we implemented our architecture in the COTSon simulator infrastructure [9], which is an appropriate tool to formalize the design of the hardware

[1] This work is partly funded by the European Commission under the projects AXIOM (645496), HiPEAC (687698)

implementation. As initial benchmarks, we use Matrix Multiplication (MM) and Recursive Fibonacci (RFib). Section 2 present the State of Art of the DataFlow execution model, Section 3 describes our system design, in the section 3 we present the MM and RFib test results in our proposed solution, varying the number of nodes and the latencies of the key operations. Finally, in Section 5 we discuss conclusions and future works

2. State of Art

The idea of a DataFlow execution model started in the 70's by Jack Dennis [3] and nowadays it is getting more attention due to the increasing overheads observed in current systems [5].
One example is ETI SWARM [4], which divides a program into tasks that can be executed when all runtime dependencies and constraints are met. Timothy et al [5] have created a framework called Open Community Runtime (OCR) to provide developers with a simplified method to make the DataFlow execution model more user-friendly. Recently, a possible implementation in heterogeneous environments has also being explored, for example in a CPU-GPU system, through the XKaapi specification [6]. Other works based on DataFlow execution model are DDM [7], TERAFLUX [8], and AXIOM [11].

3. System Design

Several problems are still open today, such as dynamic task management, hardware-engine scheduling policy, to mention a few. Our work is based on the idea of managing threads dynamically and asynchronously, using a few simple low-level key-functions. For each thread, there is a preliminary phase in which its dependencies (inputs) and outputs are specified. After that, all pertinent execution data is encapsulated in a data structure called "frame". The scheduler will deal with how, where and when to run the thread associated with the data (frame).

In the context of the AXIOM project, we are exploring the feasibility of this approach to a set of embedded boards (Fig 1), where we have several low power cores, different types of memory and FPGA hardware accelerators. We can connect multiple boards via an inexpensive high-speed interconnect (under design), in order to build a distributed system.

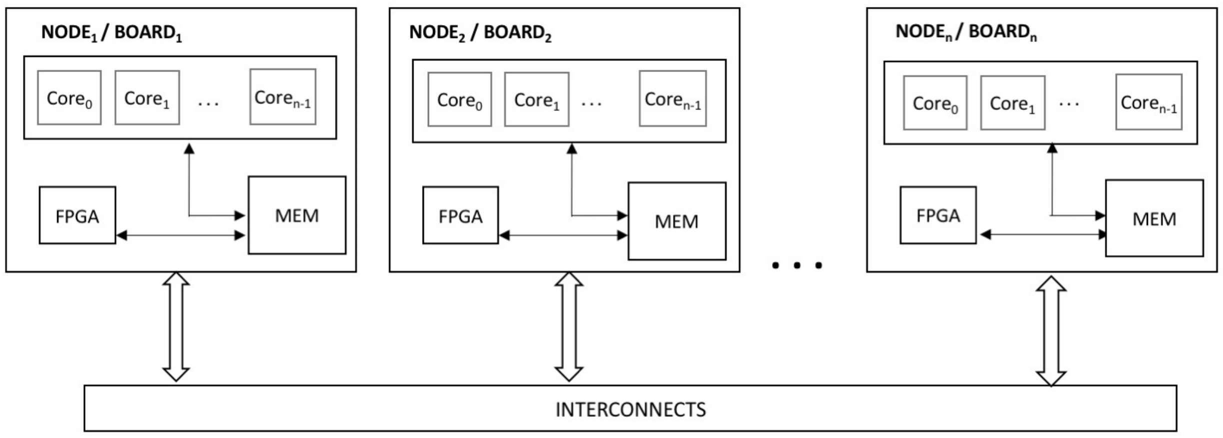
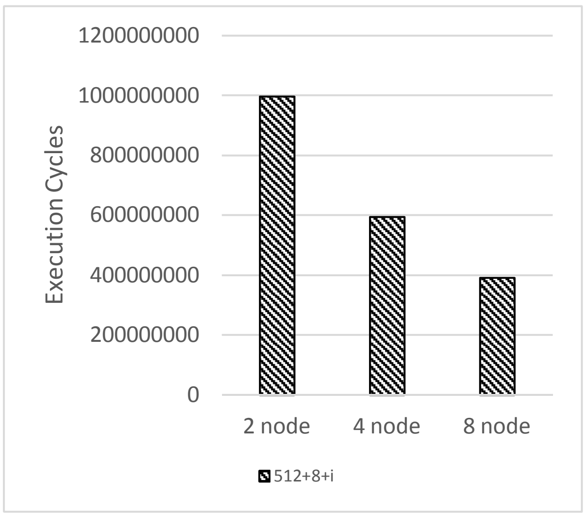
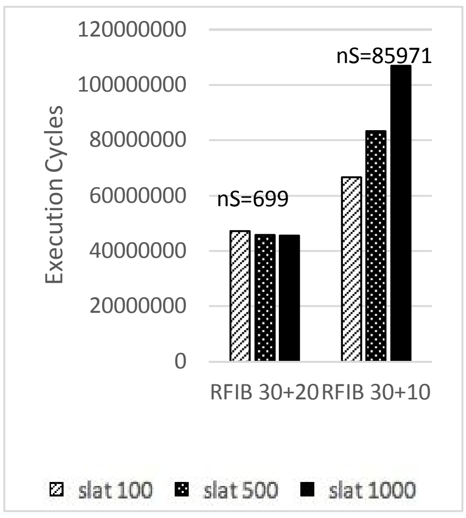


FIG 1: ARCHITECTURE OF OUT DISTRIBUTED EMBEDDED SYSTEM



(a)



(b)

FIG 2: (A) 512+8+i: MM SCALABILITY EVALUATION WITH MATRIX SIZE OF 512X512 AND BLOCK SIZE OF 8. **(B)** SCHEDULE LATENCY VARIATION (SLAT) AND ITS IMPACT ON THE RECURSIVE FIBONACCI EXECUTION CYCLES. TWO INPUT SIZE OF RFIB ARE USED, 30+20 (30 IS THE ARGUMENT OF RFIB, 20 IS THE RECURSION THREASHOLD) AND 30+10 (30 IS THE ARGUMENT OF RFIB, 10 IS THE RECURSION THREASHOLD). NS ARE THE NUMBER OF SCHEDULE OPERATIONS

4. Performance Analysis

To build up our engine for a DataFlow execution model, we used the COTSon simulator tool [9], in which we implemented both a functional and timing model. The tests were carried out with two goals:

1. Evaluate scalability changing the number of nodes of the distributed system

2. Change the latency of one of the key scheduling function to study the impact on the overall performance of a software scheduler and verify if a hardware scheduler is needed

In Fig. 2a we can notice how the system adapts its execution while increasing the number of nodes and achieving a good degree of scalability.

The experiment was executed in the COTSon simulation environment, providing a pure software implementation of the system. To improve the performance, we focused on the overhead generated by the key operations that handle the execution, looking for the solution of a more efficient hardware implementation. In this work we focused on the "schedule" operation (Fig 2b), which writes in a frame the information about a DF-Thread and returns the pointer to the related frame. We have varied its latency using 100, 500 and 1000 cycles, and we analyzed how the execution cycles impacts in case of two different input sizes for RFib. The results show that for small instances of the problem, which use few schedule operations, the impact on execution cycles is not considerable. By increasing the number of DF-threads also the schedule operations are much more frequent and the resulting overhead increases considerably.

5. Conclusion and Future Work

The attention around the DataFlow execution models is increasing: recently an IEEE STC group has been founded [13]. It seems to be the answer to the efficiency and scalability issues due to the growing demand of performance and parallelism. The proposed work shows encouraging results on scalability and speedup. Performance that can be improved by exploring innovative solutions. The impact of a huge number of schedule operation during the execution (Fig 2b) could be improved with a hardware implementation of the scheduler engine. The scalability could increase with a better exploitation of the data locality.

References

[1] Kavi, K. M., Giorgi, R., & Arul, J. (2001). Scheduled dataflow: Execution paradigm, architecture, and performance evaluation. *IEEE Transactions on Computers*, *50*(8), 834-846.

[2] Tan, X., Bosch, J., Jiménez-González, D., Álvarez-Martínez, C., Ayguadé, E., & Valero, M. (2016, April). Performance analysis of a hardware accelerator of dependence management for task-based dataflow programming models. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on* (pp. 225-234). IEEE.

[3] Dennis, J. B., Gao, G. R., & Todd, K. W. (1980). Data Flow Supercomputers. *IEEE computer*, *13*(11), 48-56.

[4] Swarm: swift adaptive runtime machine, *http://www.etinternational.com/index.php/products/swarmbeta.*

[5] Mattson, T. G., Cledat, R., Cavé, V., Sarkar, V., Budimlić, Z., Chatterjee, S., Meister, B. The Open Community Runtime: A runtime system for extreme scale computing. In *High Performance Extreme Computing Conference (HPEC), Sep. 2016 IEEE* (pp. 1-7).

[6] Gautier, T., Lima, J. V., Maillard, N., & Raffin, B. (2013, May). Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on* (pp. 1299-1308). IEEE.

[7] Kyriacou, C., Evripidou, P., & Trancoso, P. (2006). Data-driven multithreading using conventional microprocessors. *IEEE Transactions on Parallel and Distributed Systems*, *17*(10), 1176-1188.

[8] Giorgi, Roberto. "TERAFLUX: exploiting dataflow parallelism in teradevices." *Proceedings of the 9th conference on Computing Frontiers*. ACM, 2012.

[9] E. Argollo, A. Falcon, P. Faraboschi, M. Monchiero, D. Ortega, COTSon: infrastructure for full system simulation, SIGOPS Oper. Syst. Rev. 43 (1) (2009) 52–61.

[10] R. Giorgi, P. Faraboschi, "An Introduction to DF-Threads and their Execution Model", IEEE MPP, Paris, France, Oct. 2014, pp. 60-65.

[11] R. Giorgi, S. Mazumdar, S. Viola, P. Gai, S. Garzarella, B. Morelli, D. Pnevmatikatos Dionisios and Theodoropoulos, C. Alvarez, E. Ayguade, J. Bueno, D. Filgueras Antonio and Jimenez-Gonzalez, X. Martorell, "Modeling Multi-Board Communication in the AXIOM Cyber-Physical System", Ada User Journal, vol. 37, no. 4, December 2016, pp. 228-235.

[13] Parallel Model and System, dataflow and beyond, http://dfstc.capsl.udel.edu/