A Scalable Distributed Data-flow Scheduler for Many-Cores

Andrea Mondelli, Nam Ho,
Roberto Giorgi[1]

Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, University of Siena, Via Roma 56, 53100 Siena, Italy

ABSTRACT

Future exascale machines will embed 1000+ general purpose cores on a single chip.Multi-threaded applications do not exploit completely the large parallelism offered by current many-core architectures, a possible solution is to investigate more flexible execution model. This paper proposes initial results of a 1000+ cores architecture where the thread scheduling enables a dataflow execution model. Due to the huge number of threads running on the system, we introduce a way to support thread scheduling in the architecture on a many-core chip. Preliminary results on the COTSon demonstrates the ability of fully loading the target machine by distributing threads among all the available computing cores.

KEYWORDS: data-flow; simulator; scheduler; exascale;

1 Introduction

In future exascale machines, the number of cores will be one or two orders of magnitude larger than current multi-core systems. Major constraints impose serious limitations to the ability of integrating more and more processing units on the same silicon die. Reliability issues and power-wall are limiting the core number scaling. Current multi-threaded real-world applications are not capable to take advantage from this large parallelism, since the communication and synchronization overhead among threads become quickly predominant. New programming and execution models need to be designed to overcome the limitations of the current ones [SPN+09].

Dataflow paradigm is known to be capable of taking advantage of the full parallelism offered by the underlying hardware architecture, by leveraging an implicit way to eliminate data dependencies and threads synchronization [ZY12]. Although dataflow is not a new concept, only recently it has generated a revived interest.

Simulating a many-core chip architecture that embeds more than 1000 cores and efficiently scheduling threads among the computing resources is not a trivial problem [AP12].

This work presents initial results of modeling a machine based on the TERAFLUX Architecture [ea13]. To support efficient execution of threads we rely on:

[1]E-mail: {mondelli,nam,giorgi}@dii.unisi.it

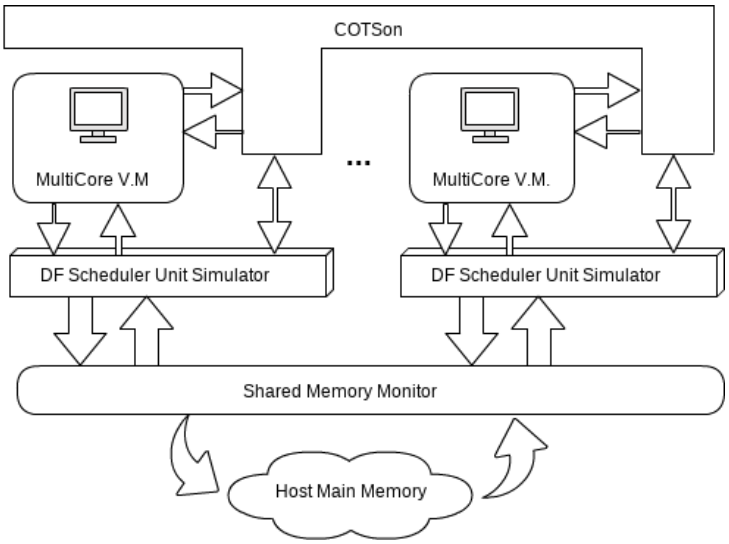

Figure 1: Simulator Setup of DF Scheduler Unit. Host memory is shared among nodes through a monitor that manage DF-Frames and Shared Frames

- a thread model which subdivides the threads in DF-Threads, Shared-Threads and Legacy/System Threads
- support for managing these threads through an X86_64 ISA extension [Gio12]
- scalable distributed DF-Scheduler architecture (See Fig.1)

Simulating a many-core chip architecture that embeds more than 1000 cores and efficiently scheduling threads among the computing resources is not a trivial problem [AP12].

2 Dataflow execution model

In our execution model, dataflow threads, called DF-threads, present a fine granularity, i.e., their body is composed of few tens of instructions. Each DF-thread is generated by the compiler to perform read accesses to the memory at the beginning of its execution, while all the write access to the memory are performed at the end of the execution. This approach allows deterministic execution. DF-Threads have no *jump* outside the thread core. This execution is DF-Driven [GPP07].

Thread dependencies are generate dinamically by the data produced bu threads themself. In order to support this producer-consumer paradigm, each DF-thread has an associated memory, called *DF-Frame* (the memory region containing all the frames is called *DF-Frame memory*).

With the aim of supporting the firing-rule, a *synchronization counter* (SC) is associated to each DF-thread. The SC is set to the number of inputs needed by the thread to become ready, and it is decremented every time a new input is produced. When SC is reduced to zero for a DF-thread, it passes from the *waiting* state to the *ready* one, as mentioned above. Finally, DF-thread creation is explicitly performed by other DF-threads during their execution, while DF-thread removal is implicitly performed at the end of the execution by freeing the associated frame.

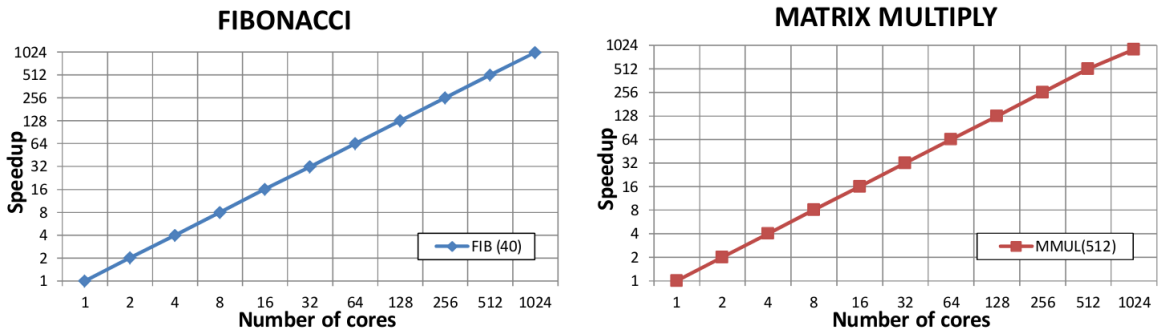

Figure 2: Speedup time increase during the execution, assuming simple $CPI = 1$ in a baseline timing model

The data-flow simulator presented in this work emulate the dataflow execution by the T-Star (T*) instructions described in [Gio12].

3 Simulator Details

The simulator presented in this paper is based on COTSon [EA09] for the off-the-shelf simulation. COTSon uses SimNow [SN09] as a virtual machine for an emulation of a full-system X86_64 architecture. As described in Figure 1, on the host machine are performed multiple SimNow instances for functional simulation of X86_64 programs , while COTSon provides decupled functional directed a timing model evaluation of the execution. The data-flow simulator proposed in this work intercepts all T* [Gio12] instructions and performs both functional behaviour and timing model. It also uses COTSon for evaluation of the standard X86_64 architecture timing model, but add to it the timing model for the hardware architecture that supports dataflow (called Thread Scheduler Unit).

The communication between different instances of SimNow is done by the use of a simulation helper called *Monitor* . This *Monitor* has direct access to the host Main Memory. In this way all memory allocations, both DF-Frame and Shared Frame [2], are handled through the *Monitor* . It allows all instances of SimNow to have access to a single memory and, thanks to the pointer returned by the *Monitor*, read or write on that memory.

4 Experimental Results and Conclusions

During the experiments, we simulated a data-flow architecture with 1024 cores. The host system used was a DLProliant DL585 G7 based on AMD Opteron 6200 Series (called TFX3), which provides 64 cores coupled to 1 TB of shared-DRAM main memory. The Fig. 2 shows the speedup of two benchmarks: fibonacci, and matrix multiplier, when we use a simple baseline model ($CPI = 1$) as done in [JWR13]. The functional simulation was launched using 64 physical cores in which each instance of SimNow simulated a 16 cores architecture. The mechanism of *Monitor* allows us to increase the number of cores simulated and the number of running SimNow instances. In addition, the centralized management of host memory allows us to have , instant by instant, a complete overview of all running T* instructions. In this way

[2] a special frame that allows sharing among nodes or cores

it is possible to synchronize the timing model for each instance of SimNow, and an overall assessment of the architecture implemented using the simulator.

These results help to investigate more in optimizing the simulator to support designing exploration. To automatize design space exploration we could also use PIKE tool [AMG13]. We realize that the T* executional model can efficiently and dynamically spawn and distribute threads. It means that if we can integrate more number of cores in a chip, we can simulate the performance of real applications. Our future works are focusing on multi-node architecture timing model which we target to tera-device chip which can scale to 1000+ cores or more.

Acknowledgment

We are very grateful to Paolo Faraboschi for his comments and useful suggestions. This work was partly funded by HiPEAC id. 287759 and the European FP7 project TERAFLUX id. 249013 http://www.teraflux.eu

References

[AMG13] C. Kang A. Mondelli and R. Giorgi. PIKE - improving COTSon interface for easier design space exploration, 2013.

[AP12] Z. Yu P. Faraboschi C. Concatto L. Carro A. Garbade S. Weis T. Ungerer and R. Giorgi A. Portero, A. Scionti. Simulating the future kilo-x86-64 core processors and their infrastructure. In *Proceedings of the 45th Annual Simulation Symposium*, ANSS '12, pages 9:1–9:7, San Diego, CA, USA, 2012. Society for Computer Simulation International.

[EA09] P. Faraboschi M. Monchiero D. Ortega E. Argollo, A. Falcón. Cotson: infrastructure for full system simulation. *SIGOPS Oper. Syst. Rev.*, 43(1):52–61, 2009.

[ea13] M. Solinas et al. The TERAFLUX project: Exploiting the dataflow paradigm in next generation teradevices. 2013.

[Gio12] R. Giorgi. TERAFLUX: exploiting dataflow parallelism in teradevices. In *Proceedings of the 9th conference on Computing Frontiers*, CF '12, pages 303–304, New York, NY, USA, 2012. ACM.

[GPP07] R. Giorgi, Z. Popovic, and N. Puzovic. Dta-c: A decoupled multi-threaded architecture for cmp systems. In *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, pages 263–270, 2007.

[JWR13] S. Yalamanchili J. Wang, Z. Dong and G. Riley. Optimizing parallel simulation of multicore systems using domain-specific knowledge. In *CM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2013.

[SN09] Amd simnow simulator 4.6.1 user's manual, 2009.

[SPN+09] K. Stavrou, D. Pavlou, M. Nikolaides, P. Petrides, P. Evripidou, P. Trancoso, Z. Popovic, and R. Giorgi. Programming abstractions and toolchain for dataflow multithreading architectures. In *Parallel and Distributed Computing, 2009. ISPDC '09. Eighth International Symposium on*, pages 107–114, 2009.

[ZY12] R. Giorgi Z. Yu, A. Righi. A case study on the design trade-off of a thread level data flow based many-core architecture, 2012.