Future Generation Computer Systems 53 (2015) 100–108

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# A scalable thread scheduling co-processor based on data-flow principles

R. Giorgi *, A. Scionti

*University of Siena, Department of Information Engineering and Mathematics, Via Roma 56, Siena, Italy*

## HIGHLIGHTS

- We present a data-flow based co-processor supporting the execution of fine-grain threads.
- We propose a minimalistic core ISA extension for data-flow threads.
- We propose a two-level hierarchical scheduling co-processor that implements the ISA extension.
- We show the scalability of the proposed system through a set of experimental results.

## ARTICLE INFO

*Article history:*
Received 31 July 2013
Received in revised form
9 October 2014
Accepted 28 December 2014
Available online 8 January 2015

*Keywords:*
Co-processor architecture
Data-flow
Many-core
High-performance systems

## ABSTRACT

Large synchronization and communication overhead will become a major concern in future extreme-scale machines (e.g., HPC systems, supercomputers). These systems will push upwards performance limits by adopting chips equipped with one order of magnitude more cores than today. Alternative execution models can be explored in order to exploit the high parallelism offered by future massive many-core chips. This paper proposes the integration of standard cores with dedicated co-processing units that enable the system to support a fine-grain data-flow execution model developed within the TERAFLUX project. An instruction set architecture extension for supporting fine-grain thread scheduling and execution is proposed. This instruction set extension is supported by the co-processor that provides hardware units for accelerating thread scheduling and distribution among the available cores. Two fundamental aspects are at the base of the proposed system: the programmers can adopt their preferred programming model, and the compilation tools can produce a large set of threads mainly communicating in a producer–consumer fashion, hence enabling data-flow execution. Experimental results demonstrate the feasibility of the proposed approach and its capability of scaling with the increasing number of cores.



## 1. Introduction

Continuous improvements in silicon technology allow the integration of an increasing number of computing units and memory space in a chip [1]. Such chips represent the elementary block for building future extreme-scale systems, but impose several challenges for designers. The International Exascale Software Project (IESP) roadmap [2] has identified the most relevant issues for, e.g., the processing elements, interconnections and memory subsystem that must be solved in order to increase 1000× in performance with relation to current high-performance systems. In order to scale, these computing systems should limit the synchronization and communication overheads among processing elements and memory subsystem [3,4]. Solving these problems may require the adoption of new programming and execution models that are able to efficiently exploit the massive parallelism offered by the underlying hardware. Since it is expected that each computing unit will manage even 100 or more concurrent threads [2,3,5], efficient and dedicated mechanisms for rapid thread context switching and synchronization will greatly help.

The TERAFLUX [6] project[1] aims at proposing technologies for improving scalability and reliability of future many-core chips. To this end, the project largely explores the data-flow paradigm (here intended in the most general way) at different levels. The capabilities of standard cores (e.g., x86_64 cores) are augmented with

* Corresponding author.
*E-mail addresses:* giorgi@dii.unisi.it (R. Giorgi), scionti@dii.unisi.it (A. Scionti).

[1] This work has been partially funded by the European FP7 project TERAFLUX id. 249013.

http://dx.doi.org/10.1016/j.future.2014.12.014
0167-739X/

hardware support for fine-grain thread scheduling. The scheduling support is exposed in the instruction set architecture (ISA) by a minimalistic set of co-processor instructions. The dedicated hardware units allow the scheduling of threads following a producer–consumer model.

This paper describes the lowest layer of the TERAFLUX hierarchy, focusing on the overall system organization, and more specifically on the hardware extension of the cores. This hardware extension is represented by a co-processor that is responsible for assisting the distribution of threads among the cores. A minimalistic set of new instructions (6 instructions) is introduced allowing the program to directly schedule new threads, read and write data from specific memory regions, and destroy threads that have completed their execution.

The rest of the paper is organized as follows. Section 2 presents relevant works regarding data-flow execution models, data-flow oriented architectures, and hardware thread schedulers. Motivations for the proposed work are explained in Section 3. Section 4 provides an overview of our system, focusing on the co-processor organization and the execution model. Section 5 introduces the instruction set extension that allows programs to support data-flow thread management. We also describe in depth two small kernel applications, to show how the co-processor directly support thread scheduling. In Section 6 we provide a description of the simulation methodology. Experimental results in Section 7 demonstrate the scalability of the proposed approach, while Section 8 concludes the paper.

## 2. Related work

The data-flow model of computation offers a simple solution to achieve high-performance by limiting the necessity of synchronization, and allowing high degree of concurrency and speculation [7,4]. However, synchronizing a large number of concurrent activities leads to sequentialization, thus defeating the potential gain of the parallel execution. In the data-flow execution model, parallel activities are enabled when they receive all the required inputs [8,9]. The model uses a graph to represent the units of computation (activities) and the flow of data among these units. Research works proposed several way to formalize this model of computation [8–12].

With the advent of many-core systems [13,14], architectures able to exploit the data-flow execution model have been proposed [15,16,6,17]. Wavescalar [18] is a tagged-token data-flow architecture designed to minimize the communication cost of moving data among processing units. The architecture implements a decentralized token-store execution model that limits the memory necessary to represent the data-flow execution graph. Scheduled DataFlow architecture [15] implements a hybrid control-/data-flow execution model. It applies the data-flow model at the thread level and decouples memory accesses from execution. Maxeler is a company specialized in designing special purpose computers based on a data-flow engine [17]. The data-flow graph is used to program reconfigurable hardware devices (FPGAs) with the aim of mapping computations to corresponding hardware functions. Etsion et al. [19] explore the data-flow execution model at the task level. Task level parallelization is supported by various programming models [20–22]. The architecture implements a hardware front-end that concurrently executes the tasks on a group of cores. Although these architectures have demonstrated to be effective in supporting the data-flow paradigm, the adoption of non standard cores and programming models limits their application.

The Kalray MPPA-256 processor [23] provides a clustered architecture similar to the one proposed in this paper. A resource management core is responsible for allocating cluster resources to the processing cores. This processor differs from our processor mainly in the adoption of Kahn process networks on top of VLIW cores, as the data-flow model.

Also other architectures resort to hardware schedulers and resource managers to gain in performance. The NVIDIA GigaThread Engine [13] issues a group of threads to each cluster of processing units based on a task allocation policy, while a local scheduler distributes the threads among the processing units. Plurality Hypercore [16] is a hybrid control-/data-flow machine. Plurality has implemented a hardware scheduler that receives the task graph description, and dispatches runnable tasks as soon as cores become available. In the next section we describe the motivations that are at the base of our work.

## 3. Motivations

With the aim of exploiting the hardware capabilities of many-core chips, programming models must evolve accordingly. Several Programming Models (PMs in the following) have been proposed to support task and thread distribution both in homogeneous and heterogeneous systems. In homogeneous systems, many-core chips presenting the same general purpose architecture (e.g., current x86_64 processors) are organized into clusters. For this kind of machines some PMs such as OpenMP [20], MPI [24], Cilk [21] became widely adopted. In heterogeneous cluster machines, the system is composed of a mix of multi-core, many-core chips and accelerators. Among the others, GP-GPUs have become widely used as accelerator platforms. PMs like CUDA [22] are designed to partition the program execution between CPUs and accelerators, and move data in and out the accelerator. With the increasing complexity of computing systems, the above mentioned PMs exhibit their limits. Synchronization among various threads running on separated cores is one of the barrier for scalability of homogeneous many-core chips [2,3]. In heterogeneous systems the PM may introduce the overhead of transferring data in and out the accelerator.

Our aim is to exploit the hardware resources by efficiently mapping programs to fine-grain data-flow threads. As of our initial experiments, data-flow programs could be automatically generated by a compilation toolchain starting from a source code written with a well-known imperative programming language (e.g., pure C/C++ language), and splitting the code in fine-grain threads (we consider the scheduled data-flow execution model) [25]. Since each thread contains few tens of instructions, the compilation toolchain can generate enough threads to maintain the machine cores utilized. Simple hardware units can be added to standard cores to effectively execute data-flow programs. By restricting the communication and synchronization to a data-flow modality, we can enable interesting properties such as isolation and repeatability of computations. Applications that could benefit from the adoption of the data-flow execution model are largely available in the High-Performance Computing (HPC) and High-Performance Embedded Computing (HPEC) domains.

## 4. System overview

One of the objectives in TERAFLUX [6,26] is the design scalability of the proposed system. Modularity [27,23] is largely used to overcome the limits of designing complex control networks, and the bottlenecks of interconnecting a huge number of processing units. Fig. 1 shows the whole system organization with a detail on the node structure. The system is organized in *nodes* interconnected by a scalable medium like a network-on-chip (NoC). I/O operations can be managed by dedicated units. Each node has a low-latency, high-bandwidth local interconnect (e.g., a shared bus, logarithmic interconnect, etc.). Internally, we assume that each node contains a dedicated memory controller (MC) and a shared last