Microprocessors and Microsystems 000 (2016) 1–15

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

On-the-fly adaptivity for process networks over shared-memory platforms

Giuseppe Tuveri[a,*], Paolo Meloni[a,*], Francesca Palumbo[b], Giovanni Pietro Seu[a], Igor Loi[c], Francesco Conti[c], Luigi Raffo[a]

[a] *DIEE, University of Cagliari, 09123 Cagliari, Italy*
[b] *PolComIng, University of Sassari, 07100 Sassari, Italy*
[c] *Micrel Lab - DEI, University of Bologna, 40136 Bologna, Italy*

ARTICLE INFO

Article history:
Received 25 November 2015
Revised 5 May 2016
Accepted 17 June 2016
Available online xxx

Keywords:
Mapping re-configuration
Adaptive MP-SoCs
Shared memory platforms
Re-configurable kahn process networks

ABSTRACT

Modern MPSoC architectures incorporate tens of processing elements on a single die. This trend poses the need of expressing the parallelism of the applications in order to effectively exploit the available resources. Several models of computation have been proposed, that specify an application as a network of independent computational elements. Such models represent a suitable solution for systematic mapping of parallel applications onto multiprocessor architectures. However, the workload of a given application can abruptly vary, as well as the amount of computing resources available, depending on the overall workload of the system and on the input data dependency. Traditional worst-case designs may overestimate workloads, leading to resource wasting and unnecessary power consumption. To overcome such limitation, in this work we devise a fast, run-time and automatic approach able to quickly re-configure the core-to-task mapping and the degree of parallelism of the application when the available resources or the application workload change, targeting shared-memory platforms. Experiments, carried out using an FPGA implementation, demonstrate the effectiveness of the proposed approach, in terms of achievable speed-up, power saving and introduced overhead.

© 2016 Published by Elsevier B.V.

1. Introduction

An increasing number of real-world applications can be classified as streaming applications. Dataflow stream-oriented models of computation like Kahn Process Networks (KPN) or Synchronous Data Flow (SDF) are suitable to specify embedded streaming applications in a parallel form, facilitating their mapping onto embedded multi-core platforms [1,2]. In such model the application tasks communicate with each other using FIFOs, thus it is often used to program statically mapped MPSoCs, where processing elements are connected by means of hardware FIFOs or point-to-point connections. However, modern MPSoC architectures incorporate a large number of processing elements whose availability may drastically change over time. For instance, if an application stops, a system manager may reallocate the released resources to another application whose performance can be improved. Several works have shown that relevant performance gains in terms of throughput [3] or energy consumption [4] can be obtained by optimizing a process network specification at compile-time. Compile-time methodologies are typically based on worst-case scenarios analysis and prevision, so that the required throughput can always be satisfied. Hence, these techniques are neither power nor resource efficient, when workloads vary at run-time. Efficient re-mapping methodologies are still an open-challenge of MPSoC design, in particular for what regard run-time task reconfiguration, according to the application need and the processing elements workload, and how the application network should be re-optimized, to dynamically allocate the proper number of cores.

In this paper we present an approach to promptly adjust the application's task mapping to cores, and to fine-tune the degree of parallelism (i.e. the capability of the overall system to execute different operations simultaneously; in a multi-core environment the degree of parallelism represents, for each time period, the number of processors working at the same time) of the application network at run-time, by means of *on-the-fly* task migration and duplication mechanisms. The approach is primarily conceived as a low-overhead technique that exploits many-core shared-memory platforms to improve efficiency of flexible communication patterns

* Corresponding author.
E-mail addresses: giuseppe.tuveri@diee.unica.it (G. Tuveri), paolo.meloni@diee.unica.it (P. Meloni), fpalumbo@uniss.it (F. Palumbo), giampiseu@gmail.com (G. Pietro Seu), igor.loi@unibo.it (I. Loi), f.conti@unibo.it (F. Conti), luigi@diee.unica.it (L. Raffo).

http://dx.doi.org/10.1016/j.micpro.2016.06.010
0141-9331/© 2016 Published by Elsevier B.V.

between cores and tasks. In particular, starting from a previous state-of-the-art work [5], in this paper:

- We propose a novel technique that allows to migrate KPN processes *on-the-fly*, while avoiding any synchronizing handshake between processes.
- We propose a novel channel-based technique that enables direct, and arbitrary, out-of-order access to the channel buffers, by multiple concurrent readers/writers while preserving the KPN semantics and avoiding any channel access serialization.
- We propose a novel run-time algorithm able to exploit the above-mentioned migration and parallelization techniques. The algorithm reacts to changes in the process workload by determining an alternative mapping that maximizes the throughput, or that minimizes the power consumption with respect to a given throughput constraint.

The rest of this paper is organized as follows. In Section 2 we provide an analysis of the state of the art, highlighting the criticalities of the current static and dynamic approaches. Section 3 provides the definition of this paper objective by means of a practical example, along with the overview of the proposed solution to dynamic, resource and power efficient, task re-mapping; while, Section 4 details all the implemented strategies. Finally, Section 5 discusses the benefits of the proposed approach, prior to conclude in Section 6 with some final remarks. Please note that different case studies are presented in Section 5 to assess and characterize our dynamic re-mapping methodology in terms of overhead with respect to a static methodology, scalability and power consumption.

2. Related work

Having an input application specified as a process network allows a more natural mapping of the application processes to the processing elements in the MPSoC architecture, with respect to a sequential program specification [1]. [6] and [7] shown that a KPN can be automatically derived from static affine nested-loop programs. Such a partitioning strategy may not necessarily result in a KPN that meets the resource or performance requirements. In [8], De Kock has shown that by modifying the network structure of a KPN, the throughput of an application can be improved. In order to meet these requirements, a designer can apply transformations to increase parallelism by splitting processes as defined in [9]. In [10], “just-enough parallelism” is exploited by replicating processes of synchronous dataflow (SDF) graphs [2]. Both approaches described in [9],[10] are performed at compile-time, by means of source-to-source compilation techniques and off-line network restructuring based on the analysis of data dependencies of the network. Working at the same level of abstraction, source-to-source transformation is meant to map one formalism/language to another; i.e. in [9] a four-step transformation methodology is presented to transform a KPN network in a functionally equivalent one, but with a changed and optimized network structure. Its Off-line restructuring is obviously less flexible with respect to a run-time adaptive approach. Moreover, the graph restructuring leads to a new network since, for each producer-consumer pair, the number of FIFOs is increased by a factor *s* (splitting or replication factor). Thus such modification is not easily applicable to the network topology at run-time (i.e. remapping FIFO ends or introducing new FIFO connections), since it would require proper middleware and hardware support to ensure data consistency. The introduction of such support can introduce a considerable overhead and requires a proper design space exploration phase, in order to find the right level of reconfiguration granularity [11,12]. Works in [9,10] are based on worst-case design methodologies, able to satisfy the throughput requirement; however, they usually are neither resource efficient nor power efficient when the workloads of nodes vary at run time [13,14].

Some approaches operating at run-time have also been proposed: [15] presents a method able to re-map tasks onto cores to improve the throughput when the workloads vary. The work in [16] performs dynamic task duplication at operating system level for streaming applications to improve the throughput. The adaptation time overhead of the use cases presented in works [15,16] is large due to coarse-grained managements, and the throughput decay lasts seconds. All of these methods are not suitable for streaming applications (e.g. audio, video and imaging) under throughput constraints with fast workload variations. Furthermore, the approach in [15] does not support any process splitting mechanism, while the approach in [16], even exploiting a run-time task duplication mechanism, performs the update of the status of each shared communication buffer only when all replicated actors (which are working on the buffer) completed their work. This can bring to a potential waste of resources. The work in [17] presents a run-time agile task mapping method for network-based many-core systems, to minimize communication latency and power dissipation, that however does not study the effects on throughput. Works in [18–20] present techniques to reduce the operating frequency/voltage of each core when workloads are smaller than the worst case, then guaranteeing the throughput constraint. Also, they allow for fast adaptation even with highly dynamic workloads. Nevertheless, the task mappings of these works are static; therefore, to optimize core usage (and in turn total power), they did not study task remapping and the possibility of reusing core across multiple applications which are running in parallel. As already said, in this paper we propose a dynamic approach, which enables run-time re-mapping capable of power consumption minimization or throughput maximization. At the state of the art, as discussed above these two key system metrics are rarely concurrently considered, and typically collide.

The methodology presented in [21] allows to adapt stateful process networks at run-time. However, it is conceived to respond to resource variations, since it requires the process network to reach a normal state in order to be transformed. This, along with a time-consuming migration technique, leads to unsuitable adaptation times for typical audio, video and imaging applications, which require fine-grained adaptation. Our approach is meant to be more lightweight in order to deal with rapid application workload variations. The work in [22] presents a novel methodology to maintain/improve the throughput of software pipelines on a many-core system, along with a fast bottleneck detection mechanism. However, it targets generic software pipelines and presents an implementation that does not support multiple tasks per core and serializes data token sending when multiple readers/writers access a buffer.

3. Context and overview of the proposed solution

Fig. 1 depicts the specification of a H.264 decoder, along with the workload distribution of the *Context-adaptive variable-length coding (CAVLC)* process across the iterations (99 process iterations per frame) for different input streams. You can notice that the workload rapidly varies at run-time, not only according to different streams, but even within each frame processing. This is due to dependency on the input data.

Having so abrupt changes in process workloads, the number of cores needed to satisfy a given throughput (e.g. 24 frames/sec) also varies at run-time. Traditional worst-case KPN compile-time approaches, such as [9] and [10], would estimate off-line network restructuring based data dependencies. Our approach, on the contrary, allows to dynamically adjust the optimal number of needed cores for a given KPN application, then freeing cores for other

Download English Version:

https://daneshyari.com/en/article/4956803

Download Persian Version:

https://daneshyari.com/article/4956803

Daneshyari.com