Microprocessors and Microsystems 54 (2017) 47–59

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Quality-of-service-aware adaptation scheme for multi-core protocol processing architecture

Mohammad Badawi[*], Zhonghai Lu, Ahmed Hemani

Department of Electronic Systems, School of ICT, KTH Royal Institute of Technology, Kista, Sweden

ARTICLE INFO

Article history:
Received 14 January 2017
Revised 4 June 2017
Accepted 21 August 2017
Available online 31 August 2017

ABSTRACT

Employing adaptable protocol processing architectures has shown a high potential in provisioning Quality-of-Service (QoS) while retaining efficient use of available energy budget. Nevertheless, successful QoS provisioning using adaptable protocol processing architectures requires adaption to be agile and to have low latency. That is, a long adaptation latency might lead to violating desired packet processing latency, desired throughput or loss of packets if the memory fails to accommodate packet accumulation. This paper presents an elastic management scheme to permit agile and QoS-aware adaptation of processing elements (PEs) within the protocol processing architecture, such that desired QoS is maintained. Moreover, our proposed scheme has the potential to reduce energy consumption since it employs the PEs upon demand. We quantify the latency required for PEs adaptation, the reduction in energy and the reduction in area that can be achieved using our scheme. We also consider two different real-life use cases to demonstrate the effectiveness of our proposed management scheme in maintaining QoS while conserving available energy.

1. Introduction

The diversity of modern telecommunication protocols and their use cases have led to an increased variation in traffic attributes, like rate and burstiness. Therefore, QoS provisioning in such use cases has become a challenge for protocol processing architectures, especially when the consumed energy needs to be conserved. However, a common practice to permit QoS provision in the case of varying traffic attributes is to dimension the protocol processing architecture based on the worst-case (traffic peak) that can occur during the execution of the application [1,2]. However, dimensioning the protocol processing architecture based on the worst-case might result in low energy efficiency as long as the worst-case is not occurring. In order to improve energy efficiency and to permit QoS provisioning, adaptable protocol processing architectures can be used [3–6].

For instance, the adaptable multi-core protocol processing architecture presented in [3] targets network nodes that have low energy-budget and are demanded to perform different roles within the network at run time. Depending on the node's role, i.e., if it is end-node or it is used to forward traffic, the protocol processing architecture used in that node experiences traffic with different attributes. To cope with this variation and to maintain energy efficiency, run-time adaption of processing cores is performed to permit different levels of parallelism. However, such adaptable protocol processing architecture (and adaptable architectures in general) can't be successfully deployed to permit QoS provisioning as long as the following requirements are not fulfilled:

1) The decision for adaptation must be evaluated with low latency.
2) Adaptation must be initiated at the right time; neither early nor late.
3) The time required to perform adaptation must be small, thus does not lead to violating the estimated worst-case values for packet processing latency and packet accumulation in memory.

In this paper we present an *Elastic Management Scheme (EMS)* that fulfills aforementioned necessary requirements and provides adequate management of the resources within the protocol processing architecture. This paper extends our previous research on the elastic management scheme [7], which is designed to provide control and management for the multi-core protocol processing architecture and the memory controller we have published in [3] and [8] respectively.

The *EMS* presented in this paper is a reconfigurable tile-based architecture. The reconfigurability of the *EMS* permits its tiles to be rapidly clustered at run-time, thus adaptation of resources can be performed. Each of these tiles contains a Static Random-Access Memory (SRAM)-based Finite-State Machine (FSM), hence called

[*] Corresponding author.
E-mail address: badawi@kth.se (M. Badawi).

http://dx.doi.org/10.1016/j.micpro.2017.08.006
0141-9331/© 2017 Elsevier B.V. All rights reserved.

FSM-tile. The FSM-tile is used to control single PE, however, if more than one PE is employed for the same use-case, an equal number of FSM-tiles can be interconnected in one cluster to control the PEs. Meaning that the main contribution of our proposed *EMS* are summarized as follows:

- It permits QoS-aware adaptation of PEs within the protocol processing architecture, such that the desired QoS can be maintained and the energy can be conserved. To make this possible we propose an approach for determining the right time to initiate adaption, where we utilize a well-established deterministic queuing theory called *Network Calculus* [9].
- It reduces the energy consumption and the latency that are required for next-state evaluation, in comparison with conventional SRAM-based FSM.
- It reduces the memory size (consequently the area) that is required when FSM-tiles are interconnecting in a cascade within a cluster, in comparison with conventional FSM-cascade approaches.

We quantify the reduction in energy, latency and SRAM area that our *EMS* can achieve and we discuss the results in Section 5. To study the effectiveness of the *EMS* in providing QoS provisioning during real-life use cases while conserving available energy, we conduct two different experiments. In the first experiment, we consider a multi-participant *Voice Over Internet Protocol (VOIP)* call to demonstrate the effectiveness of the *EMS* in providing lossless- and latency-guaranteed service. In the second experiment, we consider the case of compressing, encrypting and transmitting packets using IP Payload Compression Protocol (IPComp) [10,11] together with IP Encapsulating Security Payload IP (ESP) [12] to demonstrate the effectiveness of the *EMS* in maintaining desired throughput. The results that are collected from these two experiments are reported and discussed in detail in Section 5. In next section, we review a set of relevant research works and we discuss their similarities and differences in comparison with our proposed *EMS*.

2. Related work

In [13], Bonesana et al. proposed an adaptable architecture together with a compiler to parallelize the execution in regular expression matching. This architecture has a datapath that is organized in parallel clusters and it permits the end-designer to select the number of parallel components that fulfill the requirements of the use-case. Selecting the number of parallel components is done at compile-time using a cost function, which calculates the trade-offs between performance, power and area. Similarly to [13], our proposed design uses static analysis of the use case to permit compile-time reconfiguration. Moreover, our proposed design permits adaptation of resources at run-time, which is critical to maintain desired QoS.

In [14], Shiyanovskii et al. proposed a SW adaptation manager for run-time scheduling of reconfigurable HW fabric. The proposed adaptation manager assumes pre-configured tiles of hardware where each is tailored for certain power or performance requirement. The adaption manager has access to a library of function-configurations and it assigns application functions to the HW by loading the function-configuration that meets the demand. During execution, the adaption manager collects information related to power and processing time and use them to enrich its knowledge-base, which is used for making future adaptation decisions. Our proposed scheme also assumes a pre-configured tiles of resources to enable rapid adaptability. However, we propose a complete HW system where information regarding the processing can be collected concurrently to the main execution flow and the mechanism for making adaptation decision is embedded within the pre-configured resources.

In [15], Wu et al. proposed an elastic CPU-based architecture attempting to make the desired trade-off between power and performance through adaptation of micro-architectural resources and run-time. Our architecture is different from the one proposed in [15] since it allows the management of a multi-PE datapath that can be homogeneous or heterogeneous without restricting the PEs to be instruction-set based.

Basically, collecting processing-status and resource-status information is essential to the adaptation of resources at run-time. To collect such status and statistical information, Software (SW), Hardware (HW) or hybrid monitors can be utilized, such that QoS, buffer usage or power can be observed [16,17]. Dubach et al. in [18], proposed using HW monitors to collect system's status information at run-time then supply this information to a SW predictive model. The SW then decide to adapt the architecture, in order to fulfill performance/energy demanded during the different phases of the application. However, the authors of [18] mentioned that their method for collecting status information using the HW monitors interferes with the main execution flow and lead to a risk of saturating processing resources. Our HW monitoring mechanism is different from the once proposed in [18] since it operates concurrently without interference with the processing flow.

Ruaro et al. proposed in [17] a run-time management approach to support QoS in real-time applications. The approach proposed in [17] has some similarity to our approach where QoS thresholds are determined at compile/configuration time then run-time monitoring is used to detect surpassed thresholds. However, the scheme presented in [17] performs indirect and direct task migration between resources to utilize available and sustain QoS. With direct task migration, the task is stopped and moved (in addition to its processing context) to another resource. Alternatively, indirect task migration moves the tasks that share the resource with the task that has the priority to be served. In contrast, the resources in our approach can enabled upon demand and utilized within the execution space of the task.

Azimi et al. in [19] mentioned that HW performance monitors, not only have limited number of counters, but also their use is limited to collect micro-architectural details. As a remedy, the authors of [19] proposed using counter multiplexing as a solution. In our view, the components within the HW performance monitor and the number of needed HW counters depends on the application and the nature of events to record. However, the key requirements are the concurrency in collecting information and the low-latency notification in case of surpassed threshold(s) [20]. Our architecture fulfills these requirements since it uses HW monitors that operate concurrently to the main execution flow and have single-cycle notification latency upon surpassed threshold(s). In addition, the monitors in our proposed architecture are customizable, such that their internal components the bit-width of their components can be selected at design time according to application needs. Regardless of how it is customized, the monitor receives a continuous track of information regarding processing latency, number of executed packets and backlog size from the underlying PEs and memory controller.

In [21], García-Vergas et al. presented a *Read Only Memory (ROM)*-based implementation of the FSM. In this ROM-based implementation, the size of ROM increases with respects to the number of input events. Therefore, the authors of [21] proposed an input multiplexing mechanism, which only considers effective inputs. This input multiplexing mechanism shown a potential to reduce the size of the ROM required, however such anticipated reduction is dictated by the number of effective inputs, and in the worst-case there will be no reduction and the cost of multiplexers will be counted as an overhead. Our case assume FSMs with independent input vectors and uses a per-state state-transition table to benefit from use-case dependent improvements.