Journal of Systems Architecture 66-67 (2016) 69–83

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

An optimal allocation of memory buffers for complex multicore platforms

Andrés Goens[a], Jeronimo Castrillon[a,*], Maximilian Odendahl[b], Rainer Leupers[b]

[a] Chair for Compiler Construction, cfaed, TU Dresden, Dresden, Germany
[b] Institute for Communication Technologies and Embedded Systems, RWTH Aachen University, Aachen, Germany

ARTICLE INFO

Article history:
Received 20 March 2015
Revised 26 February 2016
Accepted 9 May 2016
Available online 10 May 2016

Keywords:
Optimization
Memory and buffer allocation
Mixed integer linear programming
Multicore
MPSoC

ABSTRACT

In deeply embedded heterogeneous multicores the allocation of data to memories is crucial for application performance. For applications with stringent throughput constraints, the allocation is often done manually by carefully assigning static memory locations to the logical buffers of the application. Today, designers are confronted with applications with thousands of buffers and architectures with hundreds of memories, rendering manual approaches impractical. In this paper we present an automatic approach for statically allocating logical buffers to physical memories, assuming a fixed task-to-processor mapping and respecting multiple throughput constraints.

In our approach, we model the application in a data-centric way, by explicitly defining buffers and associating computational tasks that access the buffers within well-specified time intervals. Besides, we use an architecture model that allows to perform an allocation that is aware of the topology of the multicore and the physical bandwidth constraints of the interconnect. We present a layered approach to describe and solve the buffer-allocation problem as well as related subproblems, using mixed-integer linear programming. We show that the buffer-allocation problem is NP-complete, and present a more scalable formulation as a semi-definite programming problem. We evaluate the proposed LP methods by allocating around 1000 buffers corresponding to processing one frame in the Long-Term Evolution (LTE) standard, onto a multicore with 80 processing elements. We introduce a solution approach that allowed to find an optimal allocation in around 2 hours, which is at least two orders of magnitude faster than a straightforward formulation.

1. Introduction

In the era of multi-cores and many-cores several programming abstractions have been proposed to hide the complexity of concurrent execution and memory management. For deeply embedded applications, however, the placement of tasks to processors and data to memories is still carefully done statically by hand. This is the case, for example, in digital baseband processing on base stations for today's wireless communication standards. In these systems, tasks are allocated statically, often to customized processors that were specifically designed to implement a particular task. Once the task allocation is done, designers spend a considerable amount of time either placing the data to the right memory or designing the memory subsystem that best suits the application. Static data allocation is preferred over a dynamic one to prevent fragmentation, non-deterministic allocation time and out-of-memory errors. Due to the increase in both software and hardware complexity, this manual allocation has become prohibitively complex, since the number of possibilities grows exponentially. The somewhat modest example of allocating 100 logical buffers to 40 physical memories already has about as many possible allocations as there are atoms in the visible universe.

The complexity of embedded software has increased as a consequence of the development of new standards (e.g., for communication or video encoding).

Applications have become more dynamic and irregular, with data and scenario-dependent execution paths. Examples of this are the different modes of the Long-Term Evolution (LTE) standard [1] or multicore engine control unit (ECU) applications in the automotive industry [2]. The hardware complexity has increased accordingly, showing a steady increase in processor counts and an even more dramatic evolution of system interconnect and memory interfaces. There are today new possibilities to interface to

* Corresponding author.
E-mail addresses: andres.goens@tu-dresden.de (A. Goens), jeronimo.castrillon@tu-dresden.de (J. Castrillon), odendahl@ice.rwth-aachen.de (M. Odendahl), leupers@ice.rwth-aachen.de (R. Leupers).

http://dx.doi.org/10.1016/j.sysarc.2016.05.002
1383-7621/© 2016 Elsevier B.V. All rights reserved.

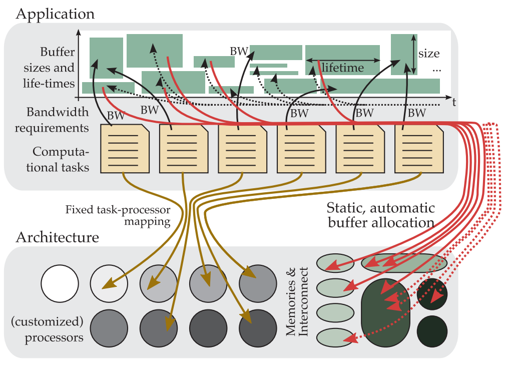

Fig. 1. A schematic example of buffer allocation.

Dynamic Random-Access Memory (DRAM) that provide high bandwidth over multiple channels. Examples for these emerging techniques, include 2.5D (High Bandwidth Memory) or 3D (Wide I/O) DRAM integration [3]. These new storage capabilities are reflected in a wide variety of communication possibilities in modern multicores. In the future, we expect even more complex architectures, such as the HAEC box [4] or the Hybrid Memory Cube (HMC) [5], where multiple paths exist to route communication between two components, such that the decision of which path to take becomes non-trivial.

The problem of mapping logical data buffers to memory has been analyzed in the context of well-structured application models with explicit communication, such as directed acyclic task graphs [6–8] and dataflow programming models [9–11]. In those models there is a clear producer-consumer relationship among tasks, which makes it easier to reason about task interactions and the impact of buffer allocation on application performance (throughput and latency). Additionally, these models only take into account the relationship between tasks that arises from the computation itself, disregarding the architecture's topology and its implications. In this paper we adopt a more general view in which logical buffers can be accessed and modified by different computational tasks at arbitrary instances of time, and which also takes into account specifics of the architecture where the computation will be executed. In our approach, instead of looking for an allocation that maximizes application throughput, we find an allocation that meets individual bandwidth demands required to meet multiple throughput constraints in complex applications.

We call the binding of a computational task running on a processor and accessing a given buffer a *flow*. Intuitively, an application is represented as a collection of small computational tasks that operate on a single shared buffer in memory. Multiple tasks can work on the same buffer, but no task accesses more than one buffer (see upper part of Fig. 1).

This model allows system architects to reason about the memory subsystem while fixing all other aspects of the execution behavior of the application.

The model used in this paper can be seen as a generalization of variable lifetime ranges in traditional compilers used for register allocation. Dataflow models, commonly used in the embedded domain [12], can also be represented with our model. The flows can be obtained from profiling runs of dataflow applications and individual bandwidth demands can be obtained from global application throughput constraints.

The *buffer-allocation* problem is then to find a *valid* and *optimal* allocation of the logical buffers of the application onto the platform memories (see Fig. 1). A valid solution is one in which all buffers can be accessed with the bandwidth required by the application and that all buffers fit into the platform memories. What is considered an optimal solution depends on the optimization criteria. In this case it is a solution that either has maximal balancing of the bandwidth loads in all channels or the balanced use of memory. A combination of both can also be considered by weighting both of the aforementioned criteria.

In this paper we introduce a mathematical model of the buffer-allocation problem and propose a solution using linear programming. The main contributions of this paper are:

- A clear presentation and formulation of the problem and a structured, layered solution using mixed-integer linear programming (MILP). This allows to decompose the problem into sub-problems and rises the potential for sequential optimizations.
- A topology-aware, optimal-allocation formulation that can deal with complex architectures and tackles memory fragmentation issues by directly generating buffer addresses.
- A detailed complexity and scalability analysis of the problem and the presented solution. In particular, we show that the problem is NP-complete.
- A more scalable formulation as a semi-definite programming problem.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 gives a formal presentation of the problem and our proposed solution with an MILP formulation. It gradually introduces sub-problems and their solutions until the complete problem is addressed. Then, an analysis of the model—including the complexity of the problem and scalability of our approach—is presented in Section 4. Section 5 presents the results of a real-world use case, namely LTE, which shows both the benefits of our approach and its limitations. Finally, we conclude our work and give an outlook on potential directions of future work in Section 6.

2. Related work

The concept of formulating and solving an allocation problem as a MILP problem is certainly not new. In the field of hardware synthesis, similar models and ideas for memory allocation have been used for application-specific integrated circuit (ASIC) design [13,14]. In more recent work, Meftali et al. [15] present a complete workflow for generating the memory subsystem, from a hardware-design perspective.

In the context of software synthesis, the problem of allocating logical memory units, like buffers, to the physical resources of a system has been considered in approaches to task-to-processor mapping [8,16,17], but only as a corollary result of the mapping. As mentioned in Section 1, allocation of communication resources for programming models based on dataflow actors and process networks have been proposed, e.g., in [11,18]. These approaches are restricted to single producer-consumer relations between computational tasks (actors or processes).

Memory address generation has also been studied in a single-processor context. Lorenz et al. [19] consider genetic algorithms for optimal performance in digital signal processors (DSPs) with single instruction multiple data (SIMD). In comparison, our memory-address generation model is much simpler, as it is only concerned with avoiding fragmentation. However, it is embedded in the rest of our approach and can thus be applied to much more complex, heterogeneous multi-processor architectures. More related to our memory-address model is the one proposed by Damavandpeyma et al. [20]. In this work, the authors devise a strategy for minimizing temperatures on scratchpad-based systems by careful variable allocation. However, their model considers a single scratchpad memory, and the objective of minimizing temperatures requires a very different allocation than in a resource-constrained multi-memory system.