Work In Progress: Design-Space Exploration of Multi-core Processors for Safety-Critical Real-Time Systems

Dolly Sapra
University Of Amsterdam
D.Sapra@uva.nl

Sebastian Altmeyer
University Of Amsterdam
altmeyer@uva.nl

***Abstract*— In this paper we outline Design Space Exploration methodology aimed at homogeneous multi-core architectures, where the safety-criticality is the crux of a system design. Multi-core architectures provide better computational abilities, but at the same time complicate the computation of timing bounds. Determining suitable architectures that achieve timing requirements is an important aspect for a system designer.**

The proposed work conceptualizes ways to automate and explore different design facets of a multi-core processor. The intention is to ensure that the particular application meets its deadlines, while optimizing other objectives such as minimizing hardware costs, energy consumption and floor area. The automated exploration builds upon Mulitcore Response Time Analysis for timing verification and multicube for heuristic search methods. The aim is to generate an architecture design in the end that can be used directly to build a custom application specific processor.

I. Introduction

With the increasing computational demand of future embedded systems, single-core processors will be replaced by modern, often application-specific multi-core designs. Multi-cores provide better performance/power ratio as well as more freedom to architect an application-specific embedded system. The assemblage of types of cores, number of cores, memory system, bus policy etc. allow the system designer to select an architecture which satisfies a range of application's design objectives. A design objective of real-time embedded systems, such as automotive or aerospace systems, is timing correctness, typically expressed via deadline requirements. In case of safety-critical systems, these requirements are strict, which means that the system do not serve its purpose if a task in the application fails to complete before its deadline.

The large design-space combined with the stringent timing requirements, poses a significant challenge to the designer of safety-critical multi-core systems: finding an application-specific multi-core design which fulfils all timing requirements, while optimizing all other design objectives, such as for instance minimizing power consumption, hardware costs, etc. In this paper, we sketch a design-space exploration for safety-critical real-time multi-core systems. We combine design-space exploration (DSE) techniques – well known already within the realm of multimedia applications – using genetic algorithms, with novel timing analysis techniques.

Ultimately, we aim to automatically create a multi-core architecture design, following the result of the DSE, that can directly be used to build a prototype FPGA processor. Such a customized application-specific processor (that ensures timing correctness of safety-critical applications) can help build a quick prototype to be used in later design stages, or even serve as an end product for low volume systems.

II. Multi-core Timing Verification

We assume an application consisting of a number of tasks, which realize different functionalities of the system. Timing analysis derives bounds on execution times, i.e., worst-case execution times (WCET) for each task. On a single core, timing bounds can be achieved through analysing execution times of tasks in isolation along with the processor scheduling policy. The WCETs of all tasks are then fed to a schedulability analysis, which computes each task's worst-case response time (WCRT), i.e., the worst-case delay between task release and task completion. If a task's WCRT is less than or equal to its deadline, the task is deemed schedulable. A task set is deemed schedulable if every task is deemed schedulable.

In a multicore system however, timing analysis is more complicated as there can be cross-core interference from shared hardware resources such as memory subsystems and common interconnects. Deriving WCETs in isolation, as commonly done for single-core system, will lead to either imprecise or optimistic execution time bounds.

Recently, a novel timing verification approach [1] for multicore systems has been developed, which combines the WCET and WCRT analysis in one framework, called the *Multicore Response Time Analysis (MRTA)*. The MRTA framework dissects the effects of the individual components of a multicore system (processor and memory demands) that contribute to the execution-time and re-assembles them at the level of the worst-case response time analysis. The analysis considers the worst-case behaviour of hardware resources, such as the memory bus, over long durations equating to task response times, rather than summing the worst case over short durations, such as single bus accesses. We build the DSE framework by extending this MRTA framework.

We represent an application with a set of n sporadic tasks $\{\tau_1, \ldots, \tau_n\}$, each task τ_i has a minimum period or inter-arrival time T_i and a deadline D_i. Deadlines are assumed to be constrained, hence $D_i \leq T_i$. The execution of task τ_i is modelled using a set of traces O_i , where each trace is an ordered list of instructions. Each instruction carries information about the memory locations accessed (if any).

We assume that the tasks are independent, i.e., they do not share mutually exclusive software resources, however, tasks compete for hardware resources such as the processor, memory, and the bus. The index of each task is unique and thus provides a global priority order, with τ_1 having the highest priority and τ_n the lowest. A subset of the tasks is assigned to each core, with local priority ordering. Furthermore, tasks follow fixed-priority preemptive scheduling.

In MRTA, the response time R_i of task τ_i on core P_x is given by the following recurrence relation:

$$R_i = \mathrm{PD}_i + I^{\mathrm{PROC}}(i,x,R_i) + I^{\mathrm{BUS}}(i,x,R_i) + I^{\mathrm{DRAM}}(i,x,R_i) \quad (1)$$

where PD_i is the processor demand, which equates to the execution time of task τ_i in isolation assuming a perfect bus and memory with zero latency. I^{PROC} is the interference on the core due to higher priority tasks preempting or delaying task τ_i. I^{BUS} is the interference on the bus computed using a mathematical model of the bus arbiter. I^{DRAM} is the interference due to DRAM refreshes. The processor demand PD, but also the memory demand MD, which denotes the number of bus accesses of a task and feeds into the computation of the I^{BUS} and I^{DRAM}, are computed by analysing the task's execution traces. The iterations start with the no-interference response time and completes either when a task's response time bound exceed the task's deadline, or when each task's response-time bound reaches a fixed-point less than or equal to the task's deadline, in which case the task set is deemed schedulable. We use this analysis to determine if a task set is schedulable on the given architecture.

The MRTA framework represents a generic and compositional solution for response time analysis. It is flexible in that it can be instantiated with various architecture design selections. It can model variable number of cores, with different arbitration policies, as well as multifarious memory models (uncached systems, data and instruction caches, scratchpads, etc., and any combination of them). We refer to [1] and [2] for details on the MRTA framework.

III. Design-Space Exploration of Safety-Critical Embedded Systems

In any embedded system, ensuring that all design objectives are met is not trivial. We can think of design space as a union of parameters and objective space (See Figure 1). Parameters are the available design choices and due to a number of possible parameter combinations, the parameter space can be exponentially large. The values of the objective space need to be acquired from relevant evaluation mechanisms, which in real-life design can not usually be trivially derived in reasonable time.

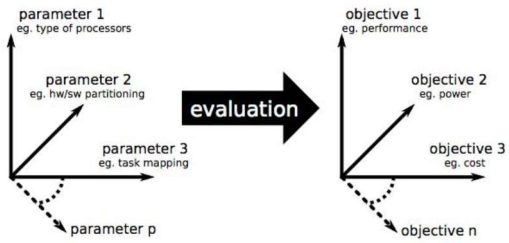

Fig. 1. Design space broken down into parameter and objective space. (taken from [3])

Most of the times, the true optimum is hard to find because there simply might not be a solution where all objectives are simultaneously met. This problem is compounded by the fact that the parameter space is too large to allow for an analysis of each point in the parameter space. For this reason, finding a design point that meets design requirements as best as possible is often considered sufficient.

Evaluating a single design point can take a considerable amount of time. Therefore, an efficient DSE should be able to use each evaluation result to either reject unsuitable parts of the design space or determine the direction of traversal in subsequent iterations. The aim is to minimize total number of evaluations needed to find a good solution (which might not be the best solution – a trade-off between time and accuracy).

DSE frameworks typically starts with a randomly chosen design point to evaluate and with each iteration attempts to move towards the optima. There are numerous heuristic algorithms available, like genetic algorithm, particle swarm optimization, simulated annealing and ant colony optimization, that govern the design space traversal by utilizing the evaluation result as feedback.

A. Application model

A given application is the starting point for this Design Space Exploration. We build our application model on the MRTA's task model, which in turn uses traces containing memory access information for each instruction. The memory information is needed to evaluate interferences based on memory demands on local memory as well as bus demands for main global memory and traces represent that information without overly complicating the analysis.

Initially we propose to use Mälardalen benchmark suite [4] to provide traces and later we aim to get traces from a real time application. The benchmarks have been compiled to the ARMv7-architecture, which also serves as the initial selection for our DSE framework.

B. Parameter Space

In our context, we model a generic homogeneous multicore architecture. Such a platform consists of l timing-compositional cores $P_1, \ldots, P_l$ as depicted in Figure 2. By timing-compositional cores we mean cores where it is safe to separately account for delays from different sources, such as computation on a given core and interference on a shared bus [5]. We assume homogenous cores, each with a local memory connected via a shared bus to a global memory and IO interface. We assume constant delays to retrieve data from global memory under the assumption of an immediate bus access. Bus transactions are atomic, which are not reordered, and non-preemptable busy waiting on the processor for requests to be serviced. Further, we assume that bus access may be given to cores for one access at a time.

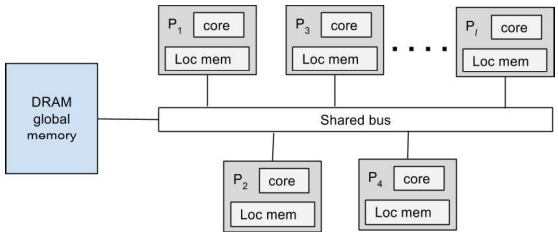

Fig. 2. Typical Multi-core platform with l cores with local memories, connected via a shared bus to the global memory.

In such a general multicore system, design parameters that can be varied and used for evaluation are

- Number of cores
- Local memory types - cache, partitioned cache and scratchpads
- Local memory, data and instruction, cache sizes
- Global memory delays
- Bus arbitration policy
- Task to core mapping

These parameters have different types of justifiable values and they need to be appropriately defined for the exploration. For example, memories can have fixed ranges, i.e. minimum and maximum allowed values which also have power of two progression. Bus arbitration policy is chosen from one of possible type like FIFO, Fixed Priority etc. The MRTA framework, which we use to evaluate individual design points is able to handle any combination of these paramters.

C. Evaluation Strategy

For an efficient exploration, we need fast and accurate evaluation mechanism for all objectives. Timing correctness comes from MRTA, which tells us if the application is schedulable or unschedulable after the analysis. Along with schedulability we need to evaluate the overall hardware cost and the floor area on an FPGA.

To have a single module that performs all evaluations, we have created a wrapper around MRTA, which accepts an architecture file, evaluates time bounds and then using simple metrics calculates the hardware cost. In future we aim to find suitable FPGA for floor plan evaluations as well as to generate an appropriate design file in the end. In exploration automated tools, objectives are also called system metrics. The tool is instantiated with configurable settings to optimize (maximize or minimize) a system metric over multiple iterations. To fasten up evaluation, we reuse partial evaluations in different iterations wherever possible and do not calculate other system metrics when a task is deemed unschedulable.

D. DSE Methodology

Incorporating all the ideas presented above, we build the whole framework as depicted in Figure 3. A full search with evaluation of each and every single point in the design space is prohibitively slow due to the size of the design space and since the analysis may take several minutes to validate the schedulability on a mutlicore design. The number of design points that can be realistically evaluated is hence limited.

We use heuristic search techniques that can be used to search the design space using a subset of design point evaluations for an optimal solution. A good search technique should be able to take feedback from evaluation results and modify its search methodology accordingly. A few automated DSE frameworks already exist, for example, Multicube [6] and FADSE [7], that can be utilised to explore the design space based on various heuristics like genetic algorithms, particle swarm optimization and simulated annealing.

We have two modules in the proposed approach – an automated design-space exploration module and an evaluation

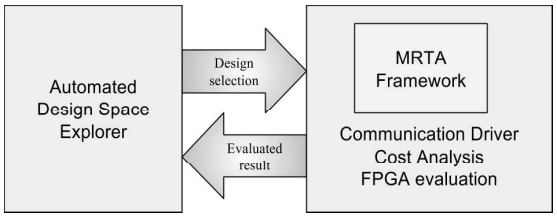

Fig. 3. Experimental design-space exploration setup

module. They communicate back and forth using predefined communication protocol. MRTA framework is used inside the wrapper module that also analyses other system metrics.

We have built the framework with Multicube explorer and the evaluation module with partial implementation of the parameter space. We still have to implement variable number of cores and dynamic task to core mapping. With the smaller parameter space, we have done full search, which is taking a long time (days) already, and we are currently working on setting up a genetic algorithm traversal.

IV. Conclutions

In this paper we presented our ongoing work towards finding appropriate multi-core architecture that meets strict timing requirements of an application. The framework contributes towards ensuring all application requirements are met, building upon automated design-space exploration and MRTA. We still need to work on covering all of parameter space, which is only partially done so far. Also, hardware cost analysis needs to be updated for realistic market values. We will evaluate heuristic algorithms, compare results and establish an efficient way to achieve the architecture design file. Future work is to integrate FPGA evaluation and synthesis into the framework to achieve a custom application specific processor.

References

[1] S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "A generic and compositional framework for multicore response time analysis," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, ser. RTNS '15, 2015, pp. 129–138.

[2] R. I. Davis, S. Altmeyer, L. S. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "An extensible framework for multicore response time analysis," *Real-Time Systems*, Jul 2017. [Online]. Available: https://doi.org/10.1007/s11241-017-9285-4

[3] M. Thompson, "Tools and techniques for efficient system-level design space exploration," Ph.D. dissertation, University of Amsterdam, 2012.

[4] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET Benchmarks: Past, Present And Future," in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, 2010, pp. 136–146.

[5] S. Hahn, J. Reineke, and R. Wilhelm, "Towards compositionality in execution time analysis: Definition and challenges," *SIGBED Rev.*, vol. 12, no. 1, pp. 28–36, Mar. 2015.

[6] C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, S. Bocchio, R. Zafalon, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, M. Wouters, C. Kavka, L. Onesti, A. Turco, U. Bondik, G. Mariani, H. Posadas, E. Villar, C. Wu, F. Dongrui, Z. Hao, and T. Shibin, "Multicube: Multi-objective design space exploration of multi-core architectures," in *2010 IEEE Computer Society Annual Symposium on VLSI*, July 2010, pp. 488–493.

[7] H. Calborean and L. Vinan, "An automatic design space exploration framework for multicore architecture optimizations," in *9th RoEduNet IEEE International Conference*, June 2010, pp. 202–207.