Evaluation of parallelism techniques on embedded multi-core platforms

May 7, 2013

Lucas Jenß
Betreuung: Prof. Dr. B. Schwarz

1 Thematic overview

As modern embedded applications evolve, their performance requirements increase. Due to the limitations of embedded platforms, such as constraints in power consumption and heat dissipation, these increased performance requirements cannot be met by increasing processor clock rates. Consequently, modern embedded processors, such as the Dual Core ARM Cortex-A9 shipped on the Xilinx Zynq platform, do not only feature more than a single processing core, but also provide support for hardware level parallelism. “Single Instruction - Multiple Data” (SIMD) techniques, which are especially well suited for media and signal processing (Kejariwal et al., 2009), are an example of hardware parallelism support. Since it is not generally possible to automatically parallelize existing applications, they need to be redesigned to take advantage of the platform enhancements.

The process of (re-)designing an application for parallelism involves finding *exploitable concurrency*, by analyzing the problem at hand, decomposing it into subproblems that can **safely execute at the same time**. Based on these subproblems, a parallel algorithm can be designed and implemented (Mattson et al., 2010). In the context of the FAUST Project (2013), a specific embedded application in need of parallelization is an obstacle detection system in cooperation with a lane guiding control (depicted in Figure 1). Evaluating the distance data measured by a laser scanner and the image stream providing extracted road marking parameters constitues a dataflow parallelization problem, for which suitable algorithms must be researched and compared. A promising approach is the use of pipeline parallelism (as provided by OpenMP), in which the computation steps are divided into “stages” that are executed on different CPU cores, thus building a “computation pipeline” (Preud’Homme et al., 2012). To further increase performance of the pipeline, the already mentioned SIMD instructions can be used.

The Go programming language was conceived, among other things, to allow for a highly productive development environment for parallel applications. While it is certainly possible to write concurrent applications in many of the programming languages widely in use today, very few of them were specifically built around the concept of concurrency (Pike, 2012). The hypothesis is thus, that by using the Go programming language to develop parallel applications on embedded platforms, one can increase developer productiveness by allowing the language to take care of tasks such as thread scheduling and memory management. Given that said features come at the cost of performance, another hypothesis that requires confirmation is that the overhead that the usage of Go introduces on an embedded platform is insignificant compared to the benefit that its usage provides.

In the course of my master seminar, these hypotheses will be evaluated by extracting performance critical elements from the mentioned FAUST application and implementing them in Go and C, using state of the art dataflow parallelization algorithms. These implementations will then be compared to their original (non-parallelized) counterparts, in terms of performance, ease of implementation and implementation complexity.

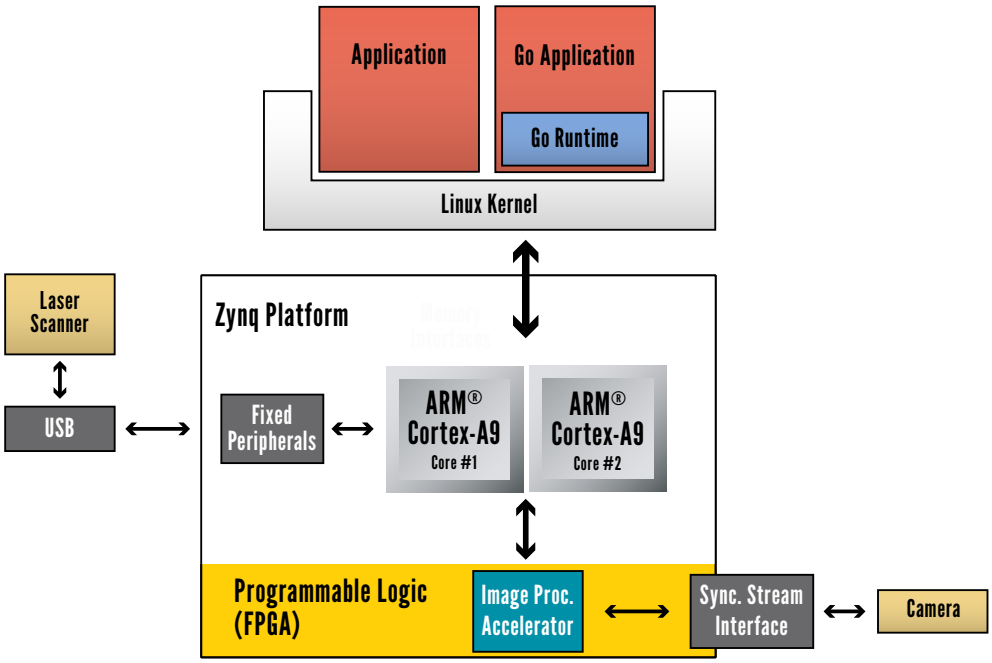

Figure 1: Obstacle detection and lane guiding control on the Zynq Platform

References

FAUST Project (2013). FAUST - Fahrerassistenz und autonome Systeme.
URL: `http://faust.informatik.haw-hamburg.de` (last accessed: 2013-05-07)

Kejariwal, A., Veidenbaum, A. V., Nicolau, A., Girkar, M., Tian, X. and Saito, H. (2009). On the exploitation of loop-level parallelism in embedded applications, *ACM Transactions on Embedded Computing Systems* **8**(2): 1–34. ISSN: 15399087.
URL: `http://portal.acm.org/citation.cfm?doid=1457255.1457257`

Mattson, T. G., Sanders, B. A. and Massingill, B. L. (2010). *Patterns for Parallel Programming*, 6th edn, Addison-Wesley, Westford, Massachusetts. ISBN: 0-321-22811-1.

Pike, R. (2012). Go at Google, *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity - SPLASH '12* p. 5.
URL: `http://dl.acm.org/citation.cfm?doid=2384716.2384720`

Preud'Homme, T., Sopena, J., Thomas, G. and Folliot, B. (2012). An Improvement of OpenMP Pipeline Parallelism with the BatchQueue Algorithm, *2012 IEEE 18th International Conference on Parallel and Distributed Systems* (i): 348–355.
URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6413677`