Going Multicore for Embedded Software

MC4ES - IWT Tetra 110174

Researchers: Van Landschoot, Silke 1 Vincke, Robbie1
Van Hove, Hugo2 Engels, Peter2
Project leaders: Boydens, Jeroen1,3 Temmerman, Marijn2
Institution(s): 1aKHBO” Dept of Industrial Engineering Science & Technology
2aKdG” Dept of Industrial Engineering Science & Technology
3aKU Leuven” Dept of Computer Science
Contact: Jeroen.Boydens@khbo.be Marijn.Temmerman@KdG.be

Multicore embedded systems introduce new opportunities and challenges. Scaling of computational power is one of the primary reasons to adopt a multicore embedded system. Yet, the challenge is to write optimal performing software, especially in an environment which lacks the aid of an Operating System. Parallel design patterns, such as Fork/Join and Thread Pool, assist to implement or migrate to a multicore embedded application.

In our experiments, a computational intensive example is implemented and thoroughly optimized. Therefore we use the Trial division algorithm which is commonly used for prime factorization in for example cryptography, more specifically RSA. Since the Trial division algorithm is a “embarrassingly parallel” problem, the theoretical maximum as defined by Amdahl’s law is almost reached. While evaluating the design patterns and application, the key performance bottlenecks are indicated. First, synchronizing access to shared variables should be reduced to a bare minimum. Next, load balancing should be optimized, to effectively scale amongst the available cores. Finally, interference from asynchronous I/O operations should be avoided at all cost.

The same method is applied to a more extended algorithm, the Fast Fourier Transform (FFT). Therefore the well known radix-2 Cooley-Tukey FFT algorithm is used. Starting from a basic singlecore implementation of the algorithm, parallel software design patterns will be applied in order to migrate to a dualcore environment. The parallel software will be developed using software tests in order to guarantee deterministic results. Different testing strategies can be applied such as fuzzing, model checking or state space exploration. Next to parallel design patterns, the algorithm will also be implemented using API’s such as OpenMP. We expect a smaller performance gain in comparison to the Trial division algorithm caused by data dependencies.

Keywords: multicore embedded software, performance scaling.