Scheduled protocol programming

Dokter, K.P.C.

Summary

With the advent of multicore processors and data centers, computer hardware has become increasingly parallel, allowing one to run multiple pieces of software at the same time on different machines. Coordination of these pieces is best expressed in a coordination language as an explicit interaction protocol that clearly defines the interactions among all components in the software.

An explicit interaction protocol not only improves code structure but also enables automated analysis of the protocol to improve execution efficiency of the software. Specifically, interaction protocols contain significant information that is essential for efficient scheduling, an activity that concerns the allocation of (computing) resources to software tasks. In this thesis, we focus in particular on improving execution efficiency through scheduling. Almost always, scheduling is the responsibility of a general-purpose operating system that makes no assumptions on the software and thereby ignores all relevant scheduling information in that software. As a result, the operating system alone cannot ensure optimally scheduled execution of the software.

In this thesis, we propose a solution that changes the protocol in the software such that it will be efficiently scheduled by the general-purpose operating system. The main idea is to take advantage of the duality between scheduling and coordination. To be precise, we analyze the protocol of the software to determine an optimal scheduling strategy for this software. Then, we enforce this optimal schedule by incorporating the strategy in the original protocol. As a result, we force the ignorant operating scheduler to follow our precomputed optimal schedule.

To achieve this larger goal, we present three smaller contributions. First, we obtain a baseline for the scheduling information that is available in a coordination language by comparing two coordination languages, BIP and Reo. Our comparison leads to the proposal of a composition operator for data-sensitive BIP architectures and the expansion of the theory of soft constraint automata with memory cells and bipolar preference values.

Next, we concretely establish all independent parts in the software (including those in the protocol) that can be scheduled. Here, we introduce two truly concurrent semantics namely multilabeled Petri nets and stream constraints in rule-based form. Using these semantics as an intermediate representation, we significantly improve state-of-the-art Reo compiler. As a byproduct, we propose a textual language called Treo that allows us to construct large protocols by composing primitive ones.

Finally, we represent all relevant scheduling information of each part of the software by expressing the software as a work automaton that expresses how much work each part of the software can/must do. We use these work automata to

develop a game-theoretic scheduling framework that formalizes scheduling as a two-player zero-sum game played on a graph. We slightly adapt existing solutions to compute a scheduling strategy for a small cyclo-static dataflow application that optimizes throughput. As promised, we enforce the optimal scheduling strategy by integrating it with the original protocol, which avoids custom changes to the default operating system scheduler.

A more extensive summary of this thesis appears in Chapter 9.