GraphML-Based Exploration and Evaluation of Efficient Parallelization Alternatives for Automation Firmware

Jürgen Bregenzer

Bosch Rexroth AG, Lohr am Main, Germany
University of Würzburg, Germany
bregenzer@informatik.uni-wuerzburg.de

1 Motivation

Graphs are an accepted and popular way of representing and solving problems of various kinds. Thus, applications of graphs as well as related topics such as graph visualization and graph representation are numerous. Our application is located in the domain of industrial automation and driven by the need of parallelizing our controller's firmware for the upcoming multi-core CPUs. As efficiency matters, we thereby aim at gaining maximum performance increases by spending no more implementation effort than necessary. In order to achieve that objective, we at first want to explore, evaluate and visualize those efficient parallelization alternatives by means of a graph-based model of our firmware. Thus, we are currently developing the *EEEEPA* (*Exploration and Evaluation of Efficient Parallelization Alternatives*) tool for this purpose. Thereby, we have chosen and extended *GraphML* [1], a widespread format for graph representation.

2 The EEEPA Tool

At first, the *EEEEPA* tool is targeting the static mapping of established tasks and interrupts to CPU cores by modeling the firmware's task system as an extended kind of a task interaction graph (TIG), the *EEEEPA.TIG*. Derived from runtime logs, the graph represents each of the firmware's tasks by a node, weighted by the task's fraction of the system's load. An *interaction edge* between two nodes indicates interaction among the corresponding tasks by means of semaphores, events and messages. As *GraphML-Attributes* are restricted to simple types, the *EEEEPA-GraphML* scheme redefines the `data-extension.type` for holding the entire interaction profile of an interaction edge as a sequence of complex types. Next, the weights of these edges are derived. This can either happen by taking hardware- and OS-specific benchmark results into account in order to consider the interaction-specific performance impacts of switching to inter-core interaction or by simply adding up the absolute interaction counts. Finally, *effort edges* are parsed from an Excel sheet. These edges indicate the development efforts of implementing proper synchronizations before running the edge-adjacent tasks on different CPU cores. The efforts are commonly estimated by developers.

Fig. 1. Visualization of parallelization alternatives: The *EEPA.TIG* (1) for a mapping of tasks and interrupts and the *EEPA.CDFG* (2) for the parallel schedule of a task's or interrupt's decomposition.

However, the sole mapping of established tasks and interrupts can cause an imbalanced core utilization due to coarse grain ones, that are consuming a significant fraction of processing time. Thus, decomposition of established tasks and interrupts is also targeted by the *EEPA* tool. The employed graph model for this purpose is a special kind of a control/data flow graph (CDFG), the *EEPA.CDFG*. Again, the graph is constructed on basis of runtime logs: By source code instrumentation, developer-defined code blocks are enclosed by specific logging events. The derived graph comprises a node for each code block, that is weighted by the code block's runtime. Nodes are connected by unweighted *control flow edges* if logging revealed a control flow between them. Weighted *interaction edges* between code blocks indicate interactions in terms of data flows. These edges are derived by means of logging events and source code annotations, that are commonly defined by developers, maybe by assistance of static code analysis. The aforementioned *GraphML* extension again provides means for holding all data flow details between code blocks. Finally, an *effort edge* is added along each control flow edge, as implementation effort is induced by control flow adaptation for running the adjacent code blocks on different cores.

In case of both graph-based firmware models, a multi-objective problem solver is engaged for exploring and evaluating mapping or scheduling alternatives, that are Pareto optimal for a given set of load profiles. For finally selecting an alternative for implementation, not solely their ratings with respect to core balance or schedule length, inter-core interaction and implementation effort are of interest: An appropriate visualization of the alternatives, that highlights and annotates parallelization-specific issues such as core-crossing edges, can be of great advantage. Figure 1 is depicting an according sketch of a visualization.

Reference

1. Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., Marshall, M.S.: GraphML Progress Report: Structural Layer Proposal. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 501–512. Springer, Heidelberg (2002)