

CAST – A Task-Level Concurrency Analysis Tool

(Extended Abstract)

Sander Stuijk, Twan Basten, and Jan Ypma

Eindhoven University of Technology, PO Box 513, NL-5600 MB, Eindhoven, The Netherlands.

s.stuijk@tue.nl

Abstract

CAST is a system-level software tool for target-architecture-independent concurrency optimization of streaming applications. It includes a design exploration method to guide a system designer in an intuitive way through the design-space.

1. Introduction

Next-generation embedded multi-media systems will often be built on multi-processor systems to obtain high compute power at relatively low energy cost. To exploit the concurrency inherently present in these systems, the parallelism available in an application mapped onto the multi-processor system must be made visible in the mapping and programming trajectory. This abstract presents a system-level software tool and accompanying design exploration method to extract task-level concurrency from an executable specification. The optimization criteria are inspired by those used in performance analysis but targeted towards streaming and concurrency. The novelty is that we perform target-architecture-independent optimization at the executable-specification (source-code) level. Using this optimization, we are able to derive a specification that can easily be optimized for many different platforms. In other words, it helps in making re-usable specifications. For details, the reader is referred to [5].

2. Models of Computation and Concurrency

Our model of computation, the computational-network model, assumes that a parallel computation is organized as a hierarchical collection of autonomous compute nodes that are connected to each other by means of point-to-point connections corresponding to data streams. A given node computes on data it receives along its inputs to produce output on some or all of its outputs. The actions performed by a

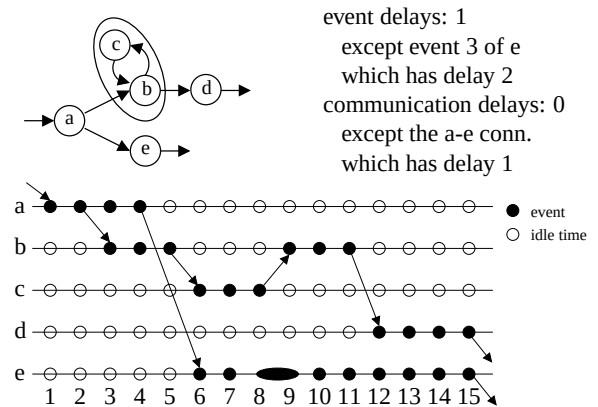


Figure 1. A network with event diagram.

node are modeled as a totally ordered sequence of events, and the actions of a network as a partial order of events. To reason accurately about timing aspects without referring to concrete implementations, we use an adapted version of Lamport's logical clocks [4], associating a delay with events and with communication. Figure 1 shows a computational network with an event diagram. The computational-network model is used to model applications for image, video and graphics processing (e.g. a JPEG decoder or a motion estimator). It captures the core of parallel (streaming) applications, and nothing more. It allows for many instantiations. In our implementation, we use a C++ implementation of Kahn Process Networks [1], namely YAPI [3]. Other models, like synchronous data-flow, are also amenable to our techniques.

Our concurrency model aims at performing a target-architecture-independent concurrency optimization. Its measures abstract from the environment in which a network operates. Five aspects of concurrency are covered. First, the *computation load* is the ratio between the time spent on computation and the time spent on computation and communication. Second, since for a concurrent computation it is important that the workload is balanced over individual nodes, workload balance is captured in the *execution load*. Third, an abstract notion of the throughput of a

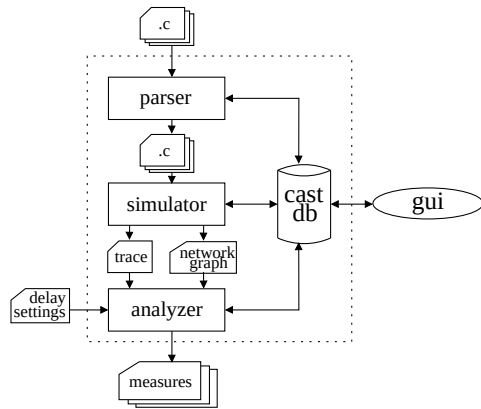


Figure 2. Overview of CAST.

computational network is provided by the *restart* measure. Fourth, the *synchronization* compares the execution time of a node in the parallel computation with the total execution time of all nodes when executed in sequence. Finally, the *structure* measure provides a notion of the number of different data streams in a network. Given a computational network and an event diagram, the five measures are easily computed. The concurrency model also provides means to identify concurrency bottlenecks. Potential bottlenecks are for example compute nodes with a poor computation load or nodes in which many data-streams come together.

3. CAST Implementation Overview

CAST can be used to compute the concurrency measures for a given computational network. The overview of CAST is shown in Figure 2. The actual computation of the measures is performed in the analyzer. The data required for this analysis is gathered through a simulation of the network. Before simulation, the parser step annotates the source code of the computational network with functions to log the execution of events during simulation. Delay settings for communication events can be provided by the designer; delays for other events, i.e., executions of C++ statements, are based on the number of instructions needed to execute these events, using a standard compiler.

Information gathered by CAST is stored in a database. This data is available for use by the graphical user interface (see Figure 3). The user interface provides a direct method of feedback on the network analysis results. The network is displayed as a graph with a customizable mapping of the measures onto the node sizes and node colors. This makes it very easy for a designer to identify potential concurrency bottlenecks. Using the hierarchy in the network, low-level details can be hidden. When a bottleneck is identified, the designer can directly jump to the corresponding C++ statement(s) in the source code. This allows for easy round-trip

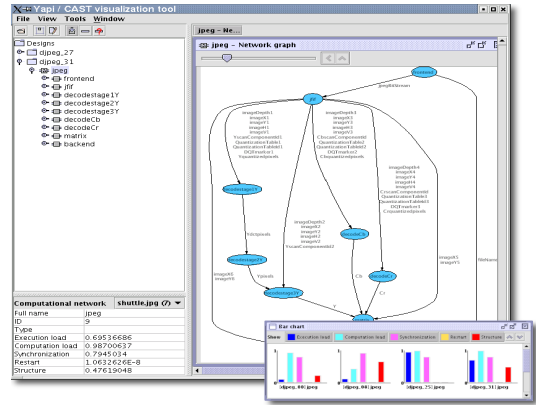


Figure 3. Screen-shot of CAST.

engineering.

CAST supports a design method for optimizing a design consisting of four steps. In the task- and data-splitting steps, all concurrency is extracted from a specification. The communication-granularity and merging steps then optimize the granularity of inter-task communication and merge tasks towards an optimal workload balance. CAST contains a statistical analysis module that can be used to minimize effects from specific simulation inputs or delay settings.

4. Results

CAST has been tested on, among others, a JPEG decoder. The performance of our solution, when mapped onto a homogeneous multi-processor platform, turned out to be similar to the performance of a JPEG decoder manually optimized for this platform [2]. The results illustrate that CAST is a versatile tool for architecture-independent analysis of task-level concurrency in streaming applications at the executable-specification level.

References

- [1] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74, Proc.*, pages 471–475. Amsterdam, The Netherlands: North-Holland, 1974.
- [2] E. A. de Kock. Multiprocessor mapping of process networks: A jpeg decoding case study. In *15th System Synthesis Symp. (ISSS)*, pages 68–73. ACM, 2002.
- [3] E. A. de Kock, et al. Yapi: Application modeling for signal processing systems. In *37th Design Automation Conf.*, pages 402–405. IEEE, 2000.
- [4] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1977.
- [5] S. Stuijk. Concurrency in computational networks. Master's thesis, TU Eindhoven, <http://www.ics.ele.tue.nl/~sander/publications/>, 2002.