Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 238 (2009) 57–69
www.elsevier.com/locate/entcs

Generating Multi-Threaded code from Polychronous Specifications

Bijoy A. Jose [a,1,2], Hiren D. Patel[b,3], Sandeep K. Shukla[a,1,4] and Jean-Pierre Talpin [c,5]

[a] *FERMAT Lab*
Virginia Polytechnic Institute and State University
Blacksburg, VA, USA

[b] *Ptolemy Group*
University of California, Berkeley
Berkeley, CA, USA

[c] *ESPRESSO Project*
IRISA/INRIA
Rennes, France

Abstract

SIGNAL, Lustre, Esterel, and a few other synchronous programming language compilers accomplish automated sequential code generation from synchronous specifications. In generating sequential code, the concurrency expressed in the synchronous programs is sequentialized mostly because such embedded software was designed to run on single-core processors. With the widespread advent of multi-core processors, it is time for model-driven generation of efficient concurrent multi-threaded code. Synchronous programming models capture concurrency in the computation quite naturally, especially in its data-flow multi-clock (polychronous) flavor. Therefore, it seems reasonable to attempt generating multi-threaded code from polychronous data-flow models. However, multi-threaded code generation from polychronous languages aimed at multi-core processors is still in its infancy. In the recent release of the Polychrony compiler, multi-threaded code generation uses micro-level threading which creates a large number of threads and equally large number of semaphores, leading to inefficiency. We propose a process-oriented and non-invasive multi-threaded code generation using the sequential code generators. By *noninvasive* we mean that instead of changing the compiler, we use the existing sequential code generator and separately synthesize some programming glue to generate efficient multi-threaded code. This paper describes the problem of multi-threaded code generation in general, and elaborates on how Polychrony compiler for sequential code generation is used to accomplish multi-threaded code generation.

Keywords: Synchronous Programming Model, Polychrony, SIGNAL, Multi-core, Multi-threading, Embedded software, Synthesis

[1] This work is supported by NSF Grant CCF-0702316
[2] Email: bijoy@vt.edu
[3] Email: hiren@eecs.berkeley.edu
[4] Email: shukla@vt.edu
[5] Email: jean-pierre.talpin@irisa.fr

doi:10.1016/j.entcs.2008.01.006

1 Introduction

In the last few years, parallel processing has claimed its niche in improving performance and power trade-off [12] for general purpose computing. So, it is no surprise that multi-core architectures will make inroads into the embedded processor markets as well. Currently, major processor vendors have already released products containing multiple cores on a single die [11], however, one must employ concurrent programming models when designing their programs to exploit the architecture with multiple cores in such products. One such programming model, and one that most designers are familiar with, is the multi-threaded programming model.

This multi-threaded programming model is commonly used in providing uniprocessor architectures with concurrency, and thus it is one candidate for true parallelism with parallel processing architectures. However, due to our long association with the von Neumann sequential programming models, it is often hard to write correct multi-threaded code [15]. Usually, writing such code involves expressing the computation as a collection of tasks, analyzing their dependencies, finding concurrency between the tasks, finding synchronization points, and then expressing all those with the programming idioms available in a language of one's choice.

We limit our discussion on multi-threaded programming for multi-core architectures with the C programming language, which to many, naturally implies the use of POSIX thread primitives or some other threading library APIs. Since writing multi-threaded C-code for a computation specified in sequential manner is hard, it would be easier if a concurrent model of computation was used instead, to specify the computation. For example, Petri nets may be used to specify the computation. With a highly concurrent Petri net model for a given computation, one could discover concurrency, synchronization points etc, much more readily when compared to standard C multi-threaded programming.

However, Petri nets have their own limitations as a specification formalism. First, one could unintentionally go very close to the implementation model in the Petri net itself, by sequentializing some transitions unnecessarily, and thereby eliminating possibility of concurrency. Figure 1 describes one such scenario where in Net 1, there is concurrency between tasks T_1 and T_2, but in Net 2, the concurrency is eliminated by ordering them. The application being modeled in this example may have had no reason to sequentialize the two tasks T_1 and T_2, except that the modeler had made a decision to do so. Also, since Petri nets are graphical formalism, often it is hard to manage for large scale programs.

An alternative to Petri nets is to use the dataflow models of computation, where the variables are considered as infinite sequences of data values, and the valuation from one step to the next is done by various operations on the data streams. The concurrency is usually difficult to sequentialize inadvertently in such specifications, because one has to explicitly impose special scheduling relations to sequentialize two operations. Polychronous languages and in particular in this paper, SIGNAL, are examples of such dataflow languages that have the notion of rate of data arrival, and multi-rate data value computation built into the language. These rates are