Thimble: Design-time Analysis of Multi-threaded System Behavior

Daniel G. Waddington
Lockheed Martin Advanced Technology Laboratories
3 Executive Campus
Cherry Hill, NJ 08002
dwadding@atl.lmco.com

Abstract

Commercial multi-core processors promise to provide more processing capacity than their single-core counterparts, in a comparable physical footprint. To fully leverage the increased processing throughput offered by multi-core solutions, however, systems must be explicitly developed with concurrent execution in mind.

Today, the predominant approach to building concurrent software is through the use of multi-threaded programming. Unfortunately, writing multi-threaded code is inherently time consuming and error prone. Furthermore, identifying the emergent behavior of multi-threaded systems before final integration and testing is almost impossible.

As part of the Lockheed Martin Software Technology Initiative (STI), we have been developing a prototype tool called Thimble to help eliminate key challenges faced by large-scale multi-threaded systems development. Using an integrated combination of modeling and run-time profiling, Thimble supports design-time exploration and understanding of the solution space, helping to both rapidly identify undesirable concurrency-related behavioral characteristics, such as lock-stepping and deadlock, and to enable "what if" analyses with respect to different possible solutions. The tool is centered on the analysis of executable models that are either directly specified or reverse engineered from legacy implementations (see Figure 1). Changes in the design can be rapidly evaluated through a visual distillation of the behavioral data.

The Thimble user builds a representative system by combining elements of program logic from existing code as well as through the use of "faux" components that mimic the behavior of components yet to be built or those that cannot run in an experimental environment. The system is modeled as a partial implementation (in our case, the implementation language is C#). Functionality directly relevant to multi-threading and task coordination (e.g., control logic) is defined in the model while other functional elements are simulated. This approach allows a precise representation, at the abstract model level, of the synchronization and concurrency semantics of the final system implementation.

Thimble executable models are implemented in the Bogor BIR guarded-transition language [2]. BIR models are directly generated from the C# source code through parsing and static program analysis. The program analysis and model generation functions are implemented in Stratego [3]. This is a functional language specifically aimed at performing AST analysis and re-writing.

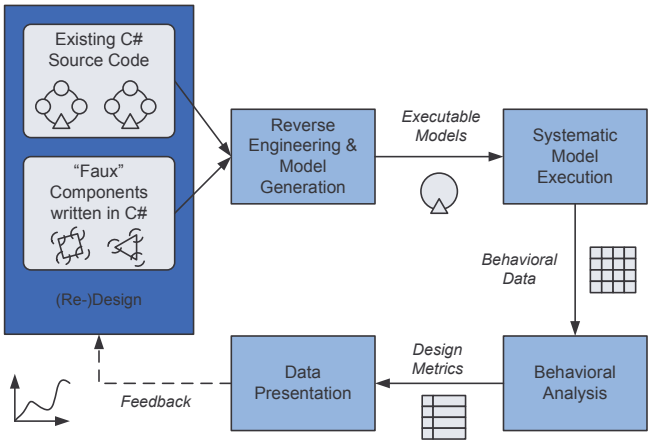

Figure 1. Logical Overview of Thimble

Once the system model is defined, the Thimble tool allows controlled execution and collection of behavioral data (for possible execution paths) with respect to scheduling patterns and synchronization interactions. Thimble collects data via a custom searcher defined within the Bogor model checking framework. Execution traces can be "mined" using a stateless randomized search or alternatively through a more guided algorithm aimed at identifying classes of behavior within the state space. Our current implementation also supports parallel execution on a multi-processor or distributed system.

Raw data from the model execution phase is analyzed offline and distilled so users can quickly identify behavioral patterns and unexpected design anomalies. Figure 2 illustrates an example Thimble graph showing the Effective Parallelism Index (EPI) of a multi-threaded data processing application over time. Intuitively, the EPI provides a measure of how threads that are created by the application are able to perform work concurrently with other threads in the system—a low EPI means that threads are highly interdependent and unable to effectively perform their work in parallel with other threads. In this example graph, there is a clear period of time (between 4900-6000 quanta) where the EPI is fluctuating between 0.5 and 1.0. This indicates

that two threads are working in lock-step with each other, which may or may not be the designer's intention.

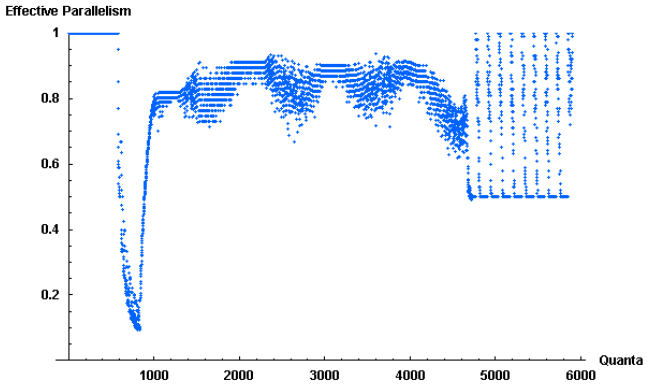

Figure 2. Example Graph of Effective Parallelism Index

The underlying objective of Thimble is to provide engineers and architects tool support to help understand the behavior of a given design. The key innovations of our solution include:

- Automated generation of executable models from existing assets through inspection of C# source code (static analysis) and collection of information from execution traces (dynamic analysis).
- Formal definitions of a number of design metrics relating to effective use of multi-threaded and multi-core architectures (e.g., Effective Parallelism Index, Saturation Index, Thread Coupling Index).
- Use of systematic model execution (partially directed model-checking) to identify both general and pathological patterns of behavior.
- Support for predictive analysis for different N-way multi-core platforms.

Our presentation will review the current Thimble prototype and present some early experimental data that demonstrates the effectiveness of our solution within the context of a large-scale satellite command and control application.

References

[1] H. Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software", Dr. Dobb's Journal, Vol. 30, March 2005.

[2] Robby, M. B. Dwyer and J. Hatcliff, "Bogor: An Extensible and Highly-Modular Model Checking Framework", in the Proceedings of the Fourth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, March 2003.

[3] E. Visser, "Stratego: A Language for Program Transformation based on Rewriting Strategies", System Description of Stratego 0.5, in Rewriting Techniques and Applications (RTA'01), Vol. 2051, Lecture Notes in Computer Science, pp. 357-361, Springer Verlag, May 2001.