Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 320 (2016) 53–67

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Bound Analysis for Whiley Programs

Min-Hsien Weng[a,1], Mark Utting[a,b,2], and Bernhard Pfahringer[a,3]

[a] *Computer Science Department Waikato University Hamilton, New Zealand*

[b] *University of the Sunshine Coast, Australia*

Abstract

The Whiley compiler can generate naive C code, but the code is inefficient because it uses infinite integers and dynamic array sizes. Our project goal is to build up a compiler that can translate Whiley programs into efficient OpenCL code with fixed-size integer types and fixed-size arrays, for parallel execution on GPUs. This paper presents an abstract interpretation-based bound inference approach along with symbolic analysis for Whiley programs. The source Whiley program is first analyzed by using our symbolic analyzer to find the matching pattern and make any necessary program transformation. Then the bound analyzer is used to analyze the transformed program to make use of primitive integer types rather than third-party infinite integer type (e.g. using GMP arbitrary precision library). The bound analysis results provide conservative estimates of the ranges of integer variables and array sizes so that efficient code can be generated and integer overflows avoided. The bound analyzer combines the bound consistency technique along with a widening operator to give fast time of solving program constraints and of converging to the fixed point. Several example programs are used to illustrate the bound analyzer algorithm and the program transformation.

Keywords: Static Analysis, Range Analysis, Abstract Interpretation, Bound Consistency, Widening Operator, Symbolic Analysis, Pattern matching, Program Transformation.

1 Introduction

Static program analysis techniques validates the consistency between software specifications and program behaviors using mathematical methodologies. For example, the bound consistency technique is widely used to solve the finite constraint domain problem (a.k.a constraint satisfaction problem)[17]. However, the problems of object-oriented program languages, such as side-effect problems or non-deterministic results, makes it a grand challenge[15] to create such a compiler, with automated

[1] Email:mw169@students.waikato.ac.nz

[2] Email: marku@waikato.ac.nz

[3] Email: bernhard@waikato.ac.nz

http://dx.doi.org/10.1016/j.entcs.2016.01.005

mathematical and logical reasoning, that can verify the specifications and detect the errors at compile-time.

Whiley[20] is a new and verification-friendly programming language with the aim of resolving verification issues that arise from object-oriented programming languages. Whiley verifying compiler can detect bugs at compile-time and convert the program into bug-less Java or C code. However, translating high-level Whiley programs into efficient implementations has some challenges, for instance, the use of unbounded integers causes substantial slowdown on the performance of Whiley implementations.

This paper aims to describe the design of bound analyzer along with symbolic analyzer to assist the code generator to produce behavior-predicable C code that makes use of efficient integer data types in the implementation. The main objectives are summarized as below:

- Recognize patterns of a Whiley program to make any necessary program transformation.
- Analyze the transformed program to produce the bound constraints.
- Infer the bounds using propagation rules and speed up bound analysis using the widening operator.
- Determine the efficient integer data types for the code generator.

This paper is organized as follows. Section 2 reviews some related works about static analysis, bound analysis and symbolic loop bounds. Section 3 describes the bound inference procedure, fixed-point approximation using widen operator, and pattern matching along with program transformation. Section 4 illustrates the algorithm of bound analysis with example programs and shows the performance of generated C code with/without program transformation. And the final section concludes the future work.

2 Related Work

2.1 Static Bound Analysis

Many automatic static program analyzers have been developed to improve the program correctness and produce the high-quality software, such as `ESC/Java Checker`[10] and `Microsoft Spec# Static Verifier`[2].

The static analysis using abstract interpretation, which approximates the abstract semantics of a computer program without executing all the calculation, allows the compiler to detect errors and find applicable optimization. For example, `Microsoft Research Clousot`[9] can statically check the absence of run-time errors and infer facts to discharge assertions.

However, computing the fixed-point in abstract domain is iterative and sometimes time-consuming. The abstract interpretation-based widening operator[6] can rely on bound results at earlier iterations, and then widen the open-ended bounds to $\pm\,\mathrm{inf}$, so as to accelerate the converging time to the fixed point. But the widening