A Compiler-based Framework for Automatic Extraction of Software Models for Exascale Hardware/Software Co-Design

Amruth R Dakshinamurthy
Modern Programming Techniques Research Lab
University of Central Florida, Orlando, FL 32816
amruth.rd@knights.ucf.edu

Damian Dechev
Scalable Computing R&D Department
Sandia National Laboratories, Livermore, CA 94550
ddechev@sandia.gov

Abstract

The utilization of large-scale parallel event simulators such as SST/macro requires that skeleton models of underlying software systems and architectures be created. Implementing such models by abstracting the designs of large-scale parallel applications requires a substantial amount of manual effort and introduces human errors. We outline an approach for automatic extraction of SST/macro skeleton models from large-scale parallel applications. Our methodology for deriving SST/macro skeleton models is based on the use of extensible and open-source ROSE compiler infrastructure. The SST/macro skeleton models are then combined with appropriate models of the network and hardware to generate wealth of information about execution pattern such as average instruction mix, memory access patterns and network utilization etc. for high-performance computing architectures. This information is useful in understanding the impact of various design decisions concerning hardware or software and will enable co-design practices to be applied to the design of future exascale systems and provide an environment to prototype ideas for future programming models and software infrastructure for these machines.

Introduction

With the growth of high-performance computing systems, understanding the behavior and performance of large-scale parallel applications on those architectures has become extremely difficult. Applications and algorithms need to constantly keep evolving to adapt to the new high-performance architectures to efficiently use the amount of parallelism offered by them. The hardware/software co-design addresses this problem through a design methodology by allowing feedback between application development and hardware design. Simulation tools such as Structural Simulation Toolkit (SST) macro simulator [1] can be extensively used that will enable hardware/software co-design process to be applied effectively to the development of high-performance computing platforms. The SST/macro simulator provides a parallel simulation environment based on MPI programming model and is driven from either a trace file or a skeleton application [1]. Trace file generation (by collecting the resulting data after executing the full application) requires high-end hardware to be available even before simulation is carried out and cannot be easily scaled to a different number of processors. The use of a skeleton application, a simplified model of the full application eliminates this problem since it is much less expensive than running the full application, yet captures the essence of the application in sufficient detail in order to generate the application's processor and network workload. Construction of skeleton models out of existing applications can be done manually, however, manual methods are time-consuming, repetitive, and run the risk of human error. Our motivation is to greatly simplify the process of creating skeleton applications by automatic extraction of skeleton models from the applications using compiler analysis techniques to significantly reduce both expenses in time and chances of error.

Key Technology Components

The key components that make up the core of our automatic translation framework as shown in Figure 1 are presented below.

The ROSE compiler framework [4] is an open source-to-source compiler infrastructure for building a wide variety of customized analysis, optimization and transformation tools. It enables rapid development of source-to-source translators from C, C++, UPC, and Fortran codes to facilitate deep analysis of complex application codes and code transformations. ROSE consists of Edison Design Group (EDG) front-ends, a midend, and backends. The front-end is used to parse C and C++ applications, midend for code analysis, optimizations and transformations, and backends to generate source code.

The SST/macro discrete event simulator [1] is an open source simulation package that enables evaluation of large-scale parallel machines. It is implemented in C++ with a modular design, permitting multiple computation and communication models to be employed. The SST/macro simulator provides several network and processor models. The network models fully support MPI and several interconnects including fat-tree, arbitrary dimension meshes and tori, and gamma graph.

The Skeleton model [1] of an application is an abstract form of the full scale application where we retain only those portions of the code (MPI calls), which are needed to determine the program flow. In other terms, a skeleton is a refactored version that is constructed by removing fragments of redundant computations and message data whose values do not affect the application's state, but retaining those code fragments specific to set of properties of interest to SST/macro simulator.

Extraction of Skeleton model

The ROSE compiler infrastructure allows programmers to construct domain-specific deep program analysis modules by allowing common forms of program analysis such as dependence analysis, control flow, and call graph etc. [4] ROSE allows for two main types of compiler extension modules: extraction modules and translation modules. Extraction modules perform program slicing by abstracting the application code and modeling only the relevant software components. This is done by identifying the values that could affect program state and to isolate the computations and communications that determine these values. The translation modules specify the desired output based on the sliced-out Abstract Syntax Tree (AST), a tree representation of the source code flow graph where each node denotes a construct occurring in the source code.

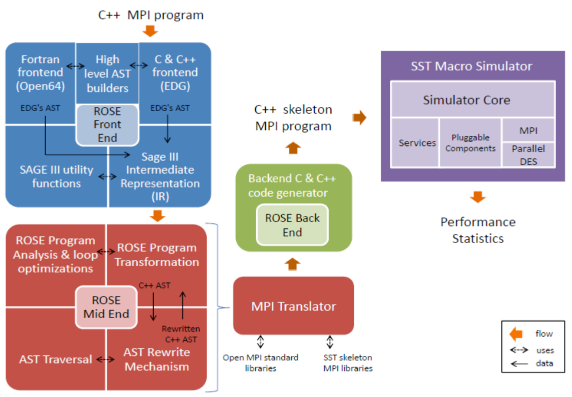

Figure 1: Automatic Translation Framework

We rely on a tool for matching arbitrary expression patterns on the AST to derive a backward slice. A backward slice is a collection of all program statements and expressions that affect a given point in the user code. Our slicing algorithm operates on the Program Dependence Graph (PDG) that is derived from the AST [4]. A PDG is a graph that models the statements as the graph's nodes and the dependencies among the program statements as directed edges. Formally defined, an edge from a statement S1 to a statement S2 exists whenever there is one or more dynamic instances in S1 that share a dependence with a later dynamic instance of S2. A PDG represents both data dependence edges and control dependence edges. A data-dependence edge from S1 to S2 indicates that the actions executed in S2 depend on the value computed in S1. A control-dependence edge from S1 to S2 models the fact that statement S2's execution is dependent on the outcome of the execution of statement S1. The implementation of the translation module is achieved by matching expression patterns against the sliced-out AST to convert the AST annotated with input application instances into AST with SST/macro-specific nodes, thus creating the skeleton model. The expression patterns (MPI calls) are those instances of the application specific to set of properties of interest to SST/macro simulator. We have selected the Jacobi relaxation algorithm as a canonical example for solving a system of linear equations using C++ [3] and MPI standard libraries to demonstrate the usefulness and accuracy of our automatic skeleton extraction methodology. The algorithm being embarrassingly parallel makes it an ideal choice for building a large-scale parallel application using MPI library targeting distributed memory architecture platforms. The C++ full scale program utilizing standard MPI calls is fed to ROSE front end. The front end parses the input source code and generates EDG's AST, the Edison Design Group's compiler intermediate representation. This AST is traversed internally to generate a new AST called Sage III Intermediate representation. The new SAGE III AST is the input to our translator that applies the program transformation and AST Rewrite/Traversal modules provided by ROSE. We apply a pre-determined traversal mechanism to find out standard MPI calls in the program and their dependencies. The ROSE compiler provides different levels of AST rewrite mechanisms for tree editing operations that can operate on statements within the AST. Each interface has only three functions: insert(), replace(), and remove() which are used by different interfaces to insert new AST fragments or to replace existing AST sub trees or to remove specific AST nodes [4]. After traversing the entire AST and editing it by matching expression patterns of standard MPI call statements using String comparisons in AST nodes, we are going to have a new rewritten AST composed of SST/macro MPI calls. We do program slicing concurrently by Top-Down-Bottom-Up traversing on the AST a final time to remove all nodes which do not contain the AstAttribute which tells us to keep the node. The backend C++ source generator uses this rewritten, sliced-out AST and unparses it to generate C++ source code which is our skeleton program. The skeleton program will only provide information about MPI calls and their associated argument lists and the transformation abstracts away all the low-level point-to-point messages that a particular MPI library uses to implement an operation. By running this skeleton program on SST/macro simulator for varying network topology and hardware layout, one can predict the modeled application's performance on large-scale parallel machines in terms of execution times.

Conclusions

Our methodology uses the program analysis capabilities of ROSE compiler to build a translator for automatically extracting skeleton models from large-scale parallel applications which use MPI programming model. The simulation results would assist in the development of high-performance computing architectures and measuring the efficiency and scalability of applications. We plan to further extend our automatic translation framework for extracting skeleton models from Sandia's Mantevo applications [5] and develop it to include other programming models.

References

[1] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, J. Mayo, "A Simulator for Large-Scale Parallel Computer Architectures," *International Journal of Distributed Systems and Technologies (IJDST)*, 1(2), 57-73. 2010.

[2] V. S. Adve, R. Bagrodia, E. Deelman, and R. Sakellariou, "Compiler-optimized simulation of large-scale applications on high performance architectures," *J. Parallel Distrib. Comput.*, 62(3):393–426, 2002.

[3] B. Stroustrup, *The C++ Programming Language*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000. *Comput.*, 62(3):393–426, 2002.

[4] ROSE Compiler, http://www.roseCompiler.org

[5] Sandia's Computational Software Site, https://software.sandia.gov/mantevo/download.html