Automatic Array Privatization *

Peng Tu and David Padua

Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign

Abstract. Array privatization is one of the most effective transformations for the exploitation of parallelism. In this paper, we present a technique for automatic array privatization. Our algorithm uses data flow analysis of array references to identify privatizable arrays intraprocedurally as well as interprocedurally. It employs static and dynamic resolution to determine the last value of a lived private array. We compare the result of automatic array privatization with that of manual array privatization and identify directions for future improvement. To enhance the effectiveness of our algorithm, we develop a goal directly technique to analysis symbolic variables in the present of conditional statements, loops and index arrays.

1 Introduction

Enhancing parallelism, balancing load and reducing communication is among the major tasks of today's parallelizing compilers. Memory-related dependence can severely limit the potential parallelism of a program. Privatization is a technique that allows each concurrent thread to allocate a variable in its private storage such that each thread accesses a distinct instance of the variable. By providing a distinct instance of a variable to each processor, privatization can eliminate memory related dependence. Previous studies on the effectiveness of automatic program parallelization show that privatization is one of the most effective transformations for the exploitation of parallelism [8]. A related technique called expansion [13] transforms each reference to a particular scalar into a reference to a vector element in such a way that each thread accesses a different vector element. When applied to an array, expansion creates a new dimension for the array.

Because the access to a private variable is inherently local, privatization reduces the communication and facilitates data distribution. Since private instances of a variable are spread among all the active processors, privatization provides opportunities to spread computation among the processors and improve load balancing [16].

* The research described was supported by Army contract DABT63-92-C-0033 and CSRD affiliate program from Motorola Corporation. This work is not necessarily representative of the positions or policies of the Army or the Government.
Previous work on eliminating memory-related dependence focused on scalar expansion [19], scalar privatization [3], scalar renaming [6], and array expansion [13] [9]. Recently there have been several papers on array privatization [11][12][16].

We present an algorithm for automatically generating an annotated parallel program from a sequential program represented by a control flow graph. In the target parallel program, each loop is annotated with its privatizable arrays and their last value assignment conditions. The algorithm has been implemented in the POLARIS parallelizing compiler. Our work on automatic array privatization presents the following new results:

- We use data flow-based analysis for array reference. Compared with the dependence analysis-based approach [12], which has to employ parametric integer programming in its most general case, our approach is more efficient and can handle nonlinear subscripts that cannot be handled by integer programming.
- The algorithm proceeds from the bottom up, which allows us to easily extend the algorithm to program call trees for interprocedural analysis. Our experience shows this interprocedural array reference analysis is necessary in many cases for successful array privatization in real applications.
- We distinguish private arrays whose last value assignments can be determined statically from those whose last values have to be assigned dynamically at runtime. This work can potentially identify more private arrays than other algorithms can identify.
- To evaluate its effectiveness, we test the algorithm on the programs in the Perfect Benchmarks. We compare the automatic privatization with manual privatization described in a previous study[8]. We find that for further improvement, more sophisticated symbolic analysis techniques are needed.
- To facilitate further improvement, we develop a goal-directed technique to analyze symbolic variables in the present of conditional statements, loops, and index arrays.

The rest of the paper is organized as follows. Section 2 is an overview of the issues in automatic array privatization and gives an example that motivates this work. Section 3 presents the algorithm. The algorithm is divided into two parts: private array identification and last-value assignment resolution. Section 4 contains the experiments of automatic privatization of the Perfect Benchmarks and presents a comparison of automatic privatization with manual privatization. Section 5 presents a goal-directed technique that uses the SSA form of a program to determine symbolic values in the presence of conditional statements, loops, and index arrays. Section 6 presents the conclusion.

2 Background

Data dependence [2] specifies the precedence constraints in the execution of statements in a program due to data producer and consumer relationships. Anti-