COMMUNICATION GENERATION FOR CYCLIC(K) DISTRIBUTIONS†

Ken Kennedy,
Nenad Nedeljkovic,
and Ajay Sethi

Department of Computer Science
Rice University, Houston, TX 77005, USA

ABSTRACT

Communication resulting from references to arrays with general cyclic(k) distributions in data-parallel programs is not amenable to existing analyses developed for block and cyclic distributions. The methods for communication generation presented in this paper are based on exploiting the repetitive nature of array accesses. We represent array access patterns as periodic sequences and use these sequences for efficient communication analysis and code generation. Our approach allows us to incorporate message coalescing optimization and to use overlap areas for shift communication. Experimental results from our prototype implementation demonstrate the validity of the proposed techniques.

1 INTRODUCTION

High Performance Fortran (HPF) is designed for portable high-level programming of parallel computers. It supports the data-parallel programming style in which Fortran 90 statements are combined with data layout directives that specify how arrays should be mapped to processors. The most general regular data distribution in HPF is the cyclic(k) (or block-cyclic) distribution; array elements are first divided into blocks of size k, and these blocks are then assigned to processors cyclically. Two major tasks in compiling programs with references to block-cyclically distributed arrays are the enumeration of local addresses for array accesses performed by each processor and the generation of interprocessor communication. In our related work, we describe efficient methods for computing each processor's local addresses for array references with arbitrary affine subscripts [7]. In this paper, we present methods for computing

†This work was supported in part by ARPA contract DABT63-92-C-0038, NSF Cooperative Agreement Number CCR-9120008, and NSF/NASA grant# ASC-9213821. The content of this paper does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.
communication sets and optimizations that improve the efficiency of the generated code.

The starting point for our work are the results by Chatterjee et al. [1]. By visualizing the layout of an array as a two-dimensional matrix, with each row containing $pk$ elements, where $p$ is the number of processors and $k$ is the block size, they show that the local memory sequence corresponding to a regular array section can be described by a finite state machine (FSM) with at most $k$ states. The FSM method is then applied to communication generation for array assignment statements. In the sending phase, each processor makes a single pass over its local data, determining the destination for each array element and building messages that consist of (address, value) pairs; message sending is followed by the execution of local iterations, after which each processor enters a receive-execute loop, receiving data items from other processors and performing non-local iterations. While this technique may lead to good data locality, especially during message packing, its disadvantage, as Stichnoth [7] points out, is that one ownership computation is required for each array access, and expensive integer divisions can incur substantial overhead.

Our techniques improve on Chatterjee et al.'s FSM approach in several ways. We take advantage of the repetitive memory access patterns to significantly reduce the number of required index translations. Using an idea previously applied in communication generation for irregular problems [2], we allocate overflow areas for non-local array elements which simplifies the generated code. Furthermore, this allows us to execute all loop iterations in the order specified by the original program. We also show how message coalescing can be applied to statements with multiple right-hand-side (rhs) references to the same array, in order to reduce the number of messages. Finally, we discuss the applicability of overlap areas, previously used only with block distributions, for a common special case of shift communication.

2 COMPUTING COMMUNICATION SETS

We first consider the following normalized loop with the arrays $A$ and $B$ having cyclic($k_A$) and cyclic($k_B$) distributions:

$$\text{do } i = 0, u$$
$$A(i \cdot s_A + l_A) = f(B(i \cdot s_B + l_B))$$
$$\text{enddo}$$

Chatterjee et al. show that for each reference, the sequence of array elements accessed by any given processor can be computed using a table of local memory gaps whose size does not exceed the block size of the array's distribution [1]. In a related work, we present an improved algorithm for constructing the table in time linearly proportional to its size [8]. The algorithm is based on finding the basis vectors ($R = (x^R, y^R)$ and $L = (x^L, y^L)$) for the integer lattice corresponding to the accessed array elements, such that for any given array element and its offset within the block to which it belongs.