Optimizing Fortran 90 Shift Operations on Distributed-Memory Multicomputers *

Ken Kennedy, John Mellor-Crummey, and Gerald Roth **

Department of Computer Science MS#132, Rice University, Houston, TX 77005-1892

Abstract. When executing Fortran 90 style data-parallel array operations on distributed-memory multiprocessors, intraprocessor data movement due to shift operations can account for a significant fraction of the execution time. This paper describes a strategy for minimizing data movement caused by Fortran 90 CSHIFT operations and presents a compiler technique that exploits this strategy automatically. The compiler technique is global in scope and can reduce data movement even when a definition of an array and its uses are separated by control flow. This technique supersedes those whose scope is restricted to a single statement. We focus on the application of this strategy on distributed-memory architectures, although it is more broadly applicable.

1 Introduction

High-Performance Fortran (HPF)[11], an extension of Fortran 90, has attracted considerable attention as a promising language for writing portable parallel programs. Programmers express data parallelism using Fortran 90 array operations and use data layout directives to direct partitioning of the data and computation among the processors of a parallel machine.

For HPF to gain acceptance as a vehicle for parallel scientific programming, it must achieve high performance on problems for which it is well suited. To achieve high performance on a distributed-memory parallel machine, an HPF compiler must do a superb job of translating Fortran 90 data-parallel operations on arrays into an efficient sequence of operations that minimize the overhead associated with data movement.

Interprocessor data movement on a distributed-memory parallel machine is typically far more costly than movement within the memory of a single processor. For this reason, much of the prior research on minimizing data movement has focused on the interprocessor case. However, although interprocessor data movement is more costly per element, the number of elements moved within the memory of a single processor may be much larger, causing the cost of local data

* This research supported in part by the NSF Cooperative Research Agreement Number CCR-9120008.
** Supported in part by the IBM Corporation through the Graduate Resident Study Program.
movement to be dominant. Johnsson previously has noted that “eliminating the local data motion by separating the set of data that must move between nodes from the data that stays within local memory may yield a significant performance improvement” [12]. The cost of local data movement becomes more important for distributed arrays as the partition size per processor increases.

In this paper, we focus on the problem of minimizing the amount of intra-processor data movement when computing Fortran 90 array operations. To make our technique as generally applicable as possible, we handle array assignment statements where the right-hand side consists of a call to a Fortran 90 shift intrinsic. The technique can handle all such assignment statements, even when the definition of the array and its uses are separated by control flow. Such a technique supersedes those that are restricted to a single statement.

In the next section we briefly review the Fortran 90 shift operators and their execution cost on distributed-memory machines. In Section 3, we describe the offset array strategy for reducing intraprocessor data movement associated with shift operations on arrays with \texttt{BLOCK} or \texttt{CYCLIC}(k) distributions. We also present some empirical results to show the potential profitability of applying the offset array optimization. Section 4 describes a global SSA-based analysis algorithm that restructures programs to use offset arrays where profitable. We close with a look at related work.

## 2 Fortran 90 Shift Operators

The Fortran 90 circular shift operator \texttt{CSHIFT(ARRAY, SHIFT, DIM)} returns an array of the same shape, type, and values as \texttt{ARRAY}, except that each rank-one section of \texttt{ARRAY} crossing dimension \texttt{DIM} has been shifted circularly \texttt{SHIFT} times. The sign of \texttt{SHIFT} determines the shift direction. The end-off shift operator \texttt{EOSHIFT(ARRAY, SHIFT, BOUNDARY, DIM)} is identical to \texttt{CSHIFT} except for the handling of boundaries. For the rest of the paper we focus on optimizing \texttt{CSHIFT} operations although our techniques can be generalized to handle \texttt{EOSHIFT} as well.

### 2.1 Sources of \texttt{CSHIFT} Operations

For HPF, optimizing \texttt{CSHIFT} operations is important since \texttt{CSHIFT} operations are ubiquitous in stencil-based dense array computations for which HPF is best suited. Besides \texttt{CSHIFT} operations written by users, compilers for distributed-memory machines commonly insert them to perform data movement needed for operations on array sections that have different processor mappings [14, 15]. For example, given the statement \texttt{X(2:255) = X(1:254) + X(2:255) + X(3:256)} the CM Fortran compiler would translate it into the following statement sequence, where the temporary arrays match the size and distribution of \texttt{X}:

\begin{verbatim}
ALLOCATE TMP1, TMP2
TMP1 = CSHIFT(X,SHIFT=-1,DIM=1)
TMP2 = CSHIFT(X,SHIFT=+1,DIM=1)
\end{verbatim}