Compiler Optimizations for Parallel Sparse Programs with Array Intrinsics of Fortran 90 *

Rong-Guey Chang
Department of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan

Tyng-Ruey Chuang
Institute of Information Science, Academia Sinica, Taipei, Taiwan

Jenq Kuen Lee
Department of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan

Abstract

Fortran 90 provides a rich set of array intrinsic functions, and they are useful to represent array expressions and data parallel programming. In the case that these intrinsic functions are applied to sparse data sets on distributed memory environments, it however is currently not supported by any vendor Fortran 90 or HPF compilers. In our research work, we attempt to provide parallel sparse supports for array intrinsics of Fortran 90. Our library uses a two-level design. In the low-level routines, it requires the input sparse matrix to be specified with compression/distribution schemes for each function. In the high-level representation, sparse array functions are overloaded for array intrinsic interfaces so that programmers need not concern about the low-level details. This raises a very interesting optimization problem in the strategies to transform the high-level representations to low-level rountines by automatic selections and supplies of distribution and compression schemes for sparse data set. In this paper, we propose solutions to address this optimization issue. Due to that the optimal problem is NP-hard, we develop a practical heuristic algorithm based on our proposed annotated graph to solve this problem. Our selection scheme, to our best knowledge, is the first work to attempt to optimize the compression and distribution schemes for sparse data array with array intrinsics on distributed memory environments. Experimental results show that our selection algorithms are consistent with our cost model, and effective in selecting the proper compression and distribution schemes to improve performances of application programs with sparse data set. Our experiments are done on IBM SP-2 with the support of our parallel sparse array intrinsics of Fortran 90.

Keywords: Parallel Sparse Compiler, Fortran 90 Array Intrinsics, Distributed Environments, Optimizing Compilers, Sparse Computations.

1 Introduction

Fortran 90 provides a rich set of array intrinsic functions. These array intrinsic functions and array expressions operate on the elements of multi-dimensional array objects concurrently without requiring iterative statements. The

*This paper is submitted to *1999 International Conference on Parallel Processing*. The correspondence author of this work is Jenq Kuen Lee. His e-mail address is jklee@cs.nthu.edu.tw, FAX number is 886-3-5723694, and phone number is 886-3-5715131 EXT. 3519.

usages of array intrinsic functions in Fortran 90 provide a rich source of parallelisms, and become more and more important due to the growing usages of Fortran 90, and the presence of the tutorial books such as *Numerical Recipes in Fortran 90*[19]. In the case that these intrinsic functions are applied to sparse data sets, however, it remains open how these intrinsic functions should be supported on both sequential and parallel systems.

In our research work, we attempt to provide parallel sparse supports for array intrinsics of Fortran 90. Our library uses a two-level design[4]. In the low-level routines, it requires the input sparse matrix to be specified with compression/distribution schemes for each function. Low-level communication routines use the Message Passing Interface (MPI) for communications of the sparse matrix among the processors. The library uses MPI and SPMD programming model. In the high-level representation, sparse array functions are overloaded for array intrinsic interfaces so that programmers need not concern about the low-level details. With our support, sparse matrix computation can now be expressed in Fortran 90 using high-level array expressions (just like dense matrix computation has always been), without concerning low-level coding of compression and distribution details. The use of two-level libraries in our design raises a very interesting optimization problem. What are the strategies to transform the high-level representations to low-level rountines?

In this paper, we address the compiler issues in transforming the high-level representations to low-level rountines by automatic selections and supplies of distribution and compression schemes for sparse data set. Previously, research work[11, 12, 5, 6, 22, 23, 14] were carried out to perform automatic data alignments and distributions mainly on dense arrays. There has been lacking research efforts to investigate the issues to automatically select distribution and compression schemes for sparse data set on distributed memory environments. Our work is distinguished from previous research work in performing automatic data alignments and distribution on dense arrays[11, 12, 5, 6, 22, 23] and previous sparse optimization work[2] for four key elements. First, the selections of distributions of sparse arrays in our work are mainly based on the sparsity of arrays, while the selections of distributions of dense arrays are mainly based on the index domain of arrays. In the selection process, the sparsity information can be given, profiled, or obtained with probabilistic inference schemes[7]. This information is important for the selection process. Second, we need to select the compression schemes for sparse arrays, and the selection was not needed on the dense cases. The compression schemes being considered are Compressed Row Storage (CRS), Compressed Column Storage (CCS), and dense representations. In addition, in our library, we extend conventional two-dimensional compression schemes (such as CRS and CCS)[25] into higher dimensional arrays. Our approach to deal with higher-dimensional arrays is to employ 1-d and 2-d sparse arrays as the bases and compose them for higher-dimension arrays. For example, we can construct a three-dimensional sparse arrays by constructing two-dimensional CRS structure with each element again a one-dimensional sparse structures. Similarly, to construct a four-dimensional arrays, one can employ 2-d sparse arrays as the bases and compose them together. The compressed structure will then be a two-dimensional compressed structure with each element again a two-dimensional sparse structure. As we are interested in providing compiler solutions to transform the high-level representations of our sparse programs to low-level rountines for Fortran 90, our compression scheme selection needs to be applicable to higher-dimensional sparse arrays. Third, our selection algorithms need to work with not only elementwise array expressions but also all sparse versions of array intrinsics in Fortran 90. These operations include CSHIFT, EOSHIFT, MATMUL, MERGE, PACK, SUM, RESHAPE, SPREAD, TRANSPOSE, UNPACK, and section moves. Many array intrinsics prefer specific representations of their input sparse arrays in terms of distribution and compression schemes for efficiency. We have incorporated this characteristic in our proposed heuristic algorithm to be shown later. Finally, we also want to perform experiments to actually compare the performances of different selected assignments of programs on distributed memory machines. Currently, this can be done for dense arrays by employing vendor's HPF compiler on distributed memory machines (such as IBM SP-2) to perform experiments for different assignments to evaluate the performances of distribution selections. This is, however, a much harder problem for sparse arrays, as currently no vendor compiler supports parallel sparse array intrinsics for Fortran 90 on distributed memory environments. Due to our research efforts and software implementation in the past two years[4], we have implemented the library to support parallel sparse computations with array intrinsics for Fortran 90 on distributed memory environments (currently is done on IBM SP-2 environments). Therefore, in our work, we actually perform experiments to compare the performances of different selected assignments of distribution and compression schemes for sparse programs on distributed memory machines.

Due to that the optimal solution of this problem is NP-hard, we have developed a cost model and proposed a practical heuristic algorithm based on our cost model and our proposed annotated graph to solve this problem. Our heuristic is done in two phases. In the first phase, we select the compression schemes, and in the second phase we select the distribution schemes. At the first phase, we first reduce the graph into a coarse graph by combining some of the neighboring nodes, together. The nodes reduced into one node are those nodes having compression schemes conforming to each other. Once the graph is reduced into a smaller graph, a tree-pruning algorithm based on the optimal solution is used on the reduced graph to perform selections for compression schemes. In the second phase, the distribution scheme is again selected based on the original annotated graph and pruning algorithms. Experimental results show that our selection algorithms are consistent with our cost model, and effective in selecting the proper compression and distribution schemes to improve performances of application programs with sparse data set. Our experiments are done on IBM SP-2 with the support of our parallel sparse array intrinsics of Fortran 90.

The remainder of the paper is organized as follows. Section 2 gives motivating examples for this work. Next, Section 3 gives the cost model and optimal solution for our problem. In Section 4, the heuristic algorithm is given. The experimental results are then given in Section 5. Finally, Section 6 give related work and Section 7 concludes this paper.

2 Motivating Examples

Fortran 90 provides a rich set of array intrinsic functions which operate on the elements of multi-dimensional array objects concurrently without requiring iterative statements. Consider the first code segment shown in Figure 1. This code segment represents the multiplication of a banded matrix and a vector. In other words, it calculates $b = Ax$, where A is a skewed representation of a banded matrix and x a vector. (Description of skewed representation and other details can be found in [19], and is omitted here.) This code uses Fortran 90 array intrinsics `eoshift`, `spread`, and a reduction function `sum`. as well as the built-in elemental (element-wise) multiplication operator, `*`. If both array A and x are dense, then the above program is expressed in a concise and data parallel style. If HPF data distribution directives are given for arrays `A`, `x`, and `b`, then presumably one can expected improved performance of the program when executed on multiprocessor machines. However, in case when arrays `A` and `b` are sparse, then such concise expression is no longer available in current Fortran 90 language. Often this means that now the same algorithm has to be coded in iterative loop over array indices. Worse, representation details about how the sparse arrays should be compressed and distributed also get mixed into the code. The resultant code is less readable and more difficult to be parallelized. The situation changes if sparse versions of the intrinsic functions (`eoshift`, `spread`, and `sum`) are available, and representation decisions of sparse matrices are hidden behind a high-level data type interface.

Figure 1 also gives another example, in a conjugate gradient solver. This code segment is translated from

Code Fragments of BandMul:

```
integer, parameter :: row = 1000
real, dimension(row,2*row-1) :: A
real, dimension(row) :: x, b
integer, dimension(2*row-1) :: shift

b = sum(A*eoshift(spread(x,dim=2,ncopies=2*row-1),&
        dim=1,shift=arth(-row+1,1,2*row-1)),dim=2)
```

Code Fragments of Conjugate Gradient Solver:

```
integer, parameter :: row =479, column = 479
double precision, dimension(row) :: x, b, r, p, Ap
double precision, dimension(row,column) :: A
double precision :: norm_r, norm_b, rtr, rtrold, beta, alpha, tol

beta = 0.0
tol = 0.1
do
  norm_r = norm(r,size(r))
  norm_b = norm(b,size(b))
  if (norm_r .le. tol*norm_b) exit
  p = r + beta*p
  Ap = MATMUL(A,p)
  alpha = rtr/DOT_PRODUCT(p,Ap)
  x = x + alpha*p
  r = r - alpha*Ap
  rtrold = rtr
  rtr = DOT_PRODUCT(r,r)
  beta = rtr/rtrold
end do
```

Figure 1: Numerical Routines with Fortran 90 Array Intrinsics

MATLAB code in [9, page 354, Fig. 8]. In the code, it uses the array intrinsic **MATMUL** for dense-matrix/dense-vector multiplication. Again, in case when arrays `A` and `b` are sparse, then such concise expression is no longer available in current Fortran 90 language.

In our research work, we attempt to provide parallel sparse supports for array intrinsics of Fortran 90. Our library uses a two-level design. In the low-level routines, it requires the input sparse matrix to be specified with compression/distribution schemes for each function. In the high-level representation, sparse array functions are overloaded for array intrinsic interfaces so that programmers need not concern about the low-level details. Our libraries are built for both sequential and parallel environments, for all sparse versions of array intrinsics in Fortran 90. These operations include CSHIFT, EOSHIFT, MATMUL, MERGE, PACK, SUM, RESHAPE, SPREAD, TRANSPOSE, UNPACK, and section moves. Below we show our sparse code fragment written with our high-level interfaces for the banded multiplication example.

```
integer, parameter :: row = 1000
type(sparse2d_real) :: A
type(sparse1d_real) :: x, b
integer, dimension(2*row-1) :: shift

call bound(A, row, 2*row-1)
call bound(x, row)
call bound(b, row)

b = sum(A*eoshift(spread(x,dim=2,ncopies=2*row-1),&
        dim=1,shift=arth(-row+1,1,2*row-1)),dim=2)
```

This is possible because Fortran 90 is a very high-level language. It allows, among others, user-defined data type, pointer access, function and operator overloading, and modular program development where implementation and interface are separated. Our library uses a two-level design. In the low-level routines, it requires the input sparse matrix to be specified with compression/distribution schemes for each function. The low-level routines are implemented by MPI and SPMD models in our implementations. In the code above, it's the high-level representation, we do not specify how the sparse matrices are compressed and distributed.

This raises a very interesting optimization problem. What are the strategies to transform the high-level representations to low-level rountines by automatic selections and supplies of distribution and compression schemes for sparse data set? In this paper, we propose schemes to address this issue. Figure 2 shows the expression tree for the banded multilication routines. T1, T2, and T3, are temporary arrays generated in the parse tree. In this paper, we will try to address the issues in selecting compression schemes and distribution mechanisms for those sparse arrays. In the case that the distribution and compression schemes of source and target arrays are already specified by programmers, we will still need to figure out the proper compression schemes and distribution mechanisms for arrays such as T1, T2, and T3, as they are temporary arrays, and will never be assigned distribution and compression schemes by programmers.

Two different assignments of compression schemes are assigned to arrays of the banded multiplication routine in Table 1 and 2, respectively, to contrast the effects of selection issues. The distribution schemes currently being considered in our problem are general block partitions based on number of non-zeroes. The compression schemes being considered are Compressed Row Storage (CRS), Compressed Column Storage (CCS), and dense representations. Note that by (Block, *) scheme, we mean distributing the sparse matrix by column in a two-dimensional array. The