Fortran 90D Intrinsic Functions on Distributed Memory Machines: Implementation and Scalability*

Ishfaq Ahmad *Rajesh Bordawekar* *Zeki Bozkus* *Alok Choudhary* *Geoffrey Fox*
Kanchana Parasuram *Ravi Ponnusamy* *Sanjay Ranka* *Rajeev Thakur*
Northeast Parallel Architectures Center, Syracuse University

Abstract

We are developing a Fortran 90D compiler, which converts Fortran 90D code into Fortran 77 plus message passing node programs for distributed memory machines. This paper presents the implementation and performance results of Fortran 90D intrinsic functions on the Intel iPSC/860 hypercube. Our implementation is portable and scalable.

1 Introduction

Fortran 90D is a version of Fortran 90 enhanced with a set of data decomposition specifications which indicate how arrays should be distributed among the processors of the distributed memory computer and also how they should be aligned with respect to one another, both within and across array dimensions. Fortran 90D is a data parallel language which can be efficiently implemented on both SIMD as well as MIMD machines. We are developing a compiler which converts Fortran 90D to Fortran 77 plus message passing node programs for a distributed memory MIMD machine [3]. In order to make the programs portable, we are implementing all the communication using EXPRESS, a portable parallel programming environment developed by Parasoft Corporation. Thus, the same source program can run without modification on a variety of machines such as Intel iPSC/2, iPSC/860 and Touchstone Delta, NCUBE and a network of workstations.

Since Fortran 90 is oriented towards scientific applications, many frequently required primitives (which operate on arrays) are provided as part of the language itself so that the user does not have to code them afresh. These are known as *intrinsic functions*. For the conversion of Fortran 90D to Fortran 77 plus message passing, it is necessary to build a library of intrinsic functions which can be called from the node programs of a distributed memory machine. This paper discusses the implementation and scalability of several of these intrinsic functions. Because of space constraints, we describe the implementation of only a few representative intrinsics and although we have performance results on iPSC/2, iPSC/860, NCUBE and a network of workstations, we give only the performance on iPSC/860. A more detailed discussion can be found in [1].

*This work was sponsored by DARPA under contract # DABT63-91-C-0028. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

2 Fortran 90D Intrinsic Functions

The intrinsic functions that we have implemented fall into four main categories :-

- *Array Reduction Functions*: ALL, ANY, COUNT, MAXVAL, MINVAL, PRODUCT, SUM.
- *Array Manipulation Functions*: CSHIFT, EOSHIFT, TRANSPOSE.
- *Array Location Functions*: MAXLOC, MINLOC.
- *Vector and Matrix Multiplication Functions*: DOT_PRODUCT, MATMUL.

For each of these functions, we have written Fortran 77 routines which can be called from the node programs of a distributed memory machine. The Fortran 90D compiler will detect calls to intrinsic functions in the Fortran 90D program and replace them with calls to these routines.

Our implementation is not specific to any particular architecture. The programmer has to specify the problem decomposition and we use EXPRESS to perform a mapping from the problem domain to the underlying processor topology. This provides the programmer with a transparent view of the underlying architecture. The programmer can assume that the processors are configured as an n-dimensional grid corresponding to the decomposition of the problem. The actual architecture of the parallel machine is effectively hidden, which makes programming easier and portable.

3 Array Reduction Functions

We describe the implementation of MAXVAL. All array reduction functions have been implemented in a

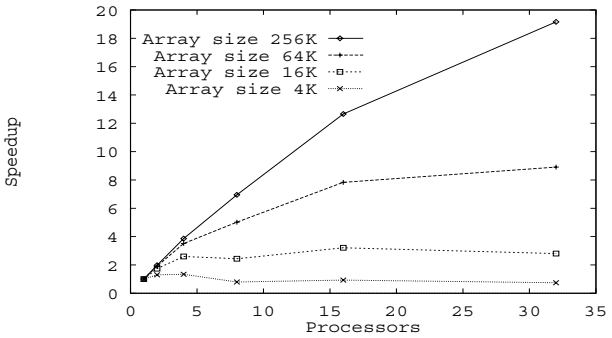

Figure 1: MAXVAL on iPSC/860

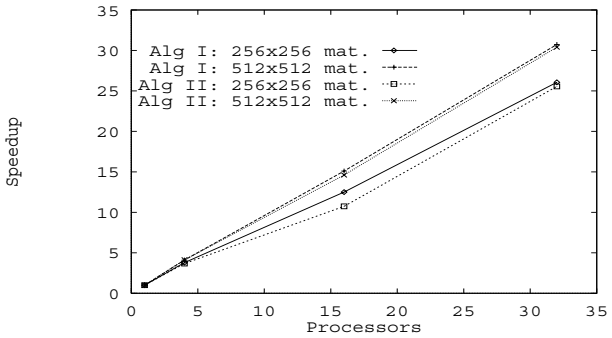

Figure 2: MATMUL on iPSC/860

similar manner. MAXVAL can be used to either find the single maximum value in the entire array (result is scalar) or to find the maximum along a particular row or column of the array (result is a vector). If the maximum value of the entire array is to be determined, each processor calculates the maximum value in the local array and then all processors perform a global maximum operation to find the maximum element among all local arrays. If the maximum value is to be found along a particular row (or column) then each processor first determines the maximum value along each row (or column) of the local array. If the rows (or columns) are distributed, then those processors which share a particular row (or column) of the array, perform a global maximum operation to determine the maximum value along the row (or column). The performance of MAXVAL on iPSC/860 for one dimensional arrays is shown in Figure 1. We see that the speedup is higher for large array sizes. This is because as the number of processors increases, the computation time decreases almost linearly, but the communication time increases. For large array sizes, the computation time is much higher than the communication time and hence as the number of processors increases, the speedup increases.

4 Matrix Multiplication

We have implemented two algorithms for MATMUL – by Fox et al [4] and Berntsen [2]. The former method requires a particular multiplicand submatrix to be broadcast to all the processors which are in the same row of the processor configuration, followed by multiplication and neighbor communication along the columns. In the latter method, initially matrices are redistributed such that only neighbor communication is necessary in subsequent steps. If the processor configuration is not a perfect square, virtual processors are created so that each processor gets a square sub-matrix. The current implementation assumes that both matrices are block decomposed and the number of processors allocated along a dimension of the array must be a multiple of the other. The performance of the two algorithms on iPSC/860 is shown in Figure 2.

The speedup increases almost linearly with the number of processors. In some cases the speedup is superlinear because of cache misses in the single processor case. The communication cost is marginally less for the second algorithm for some of the processor configurations.

5 Conclusion

We have implemented several Fortran 90D intrinsic functions so that they can be called from the node programs of a distributed memory machine. Our implementations are scalable, portable and architecture independent. They can be run without modification on a variety of machines such as Intel iPSC/2, iPSC/860 and Touchstone Delta, NCUBE and network of workstations.

References

[1] Ahmad, I., A. Choudhary, G. Fox, K. Parasuram, R. Ponnusamy, S. Ranka, and R. Thakur, *Implementation and Scalability of Fortran 90D Intrinsic Functions on Distributed Memory Machines*, Technical Report SCCS-256, March 92.

[2] Berntsen. J, *Communication Efficient Matrix Multiplication on a Hypercube*, Parallel Computing, 1989, pp 335-342.

[3] Choudhary, A., G. Fox, S. Ranka, S. Hiranandani, K. Kennedy, C. Koelbel, C. Tseng, *Compiling Fortran 77D and 90D for MIMD Distributed-Memory Machines*, in Proc. of Frontiers '92.

[4] Fox, G., S. Otto, and A. Hey, *Matrix Algorithms on a Hypercube I: Matrix multiplication*, Parallel Computing, 1987, pp. 17-31.