Fortran D Language Specification

Geoffrey Fox
Seema Hiranandani
Ken Kennedy
Charles Koelbel
Ulrich Kremer
Chau-Wen Tseng
Min-You Wu

CRPC-TR 90079
December 1990

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

FORTRAN D Language Specification[‡]

Geoffrey Fox[*]
Seema Hiranandani[†]
Ken Kennedy[†]
Charles Koelbel[†]
Ulrich Kremer[†]
Chau-Wen Tseng[†]
Min-You Wu[*]

October 17, 1993

Abstract

This paper presents FORTRAN D, a version of FORTRAN enhanced with data decomposition specifications. It is designed to support two fundamental stages of writing a data-parallel program: *problem mapping* using sophisticated array alignments, and *machine mapping* through a rich set of data distribution functions. We believe that FORTRAN D provides a simple machine-independent programming model for most numerical computations. We intend to evaluate its usefulness for both programmers and advanced compilers on a variety of parallel architectures.

1 Introduction

Recent advances in large-scale parallel processing technology has made much computing power available for today's computation scientists and engineers. However, the usefulness of parallel processing has been limited by the difficulty of programming the great variety of rapid evolving parallel architectures. A major goal for the Center for Research on Parallel Computation (CRPC) is to develop a machine-independent parallel programming model usable for both shared and distributed-memory SIMD/MIMD architectures.

High-level parallel languages such as Linda [CG89], Strand [FT90, FO90], and Delirium [LS91] are valuable when used to *coordinate* coarse-grained functional parallelism. However, these languages do not meet the needs of computational scientists because they are inefficient for capturing fine-grain data parallelism (of the type described by Hillis and Steele [HS86] and Karp [Kar87]). This is mainly due to the fact that existing parallel languages lack both language and compiler support to assist in efficient data placement [PB90]. Parallelism must also be explicitly specified because these languages do not provide compilers that can automatically detect and exploit parallelism.

---

[*] NPAC, Syracuse University, Syracuse NY 13244

[†] Department of Computer Science, Rice University, Houston TX 77251

[‡] This research was supported by the Center for Research on Parallel Computation, a National Science Foundation Science and Technology Center.

To overcome this deficiency, we have designed FORTRAN D, a version of FORTRAN enhanced with a rich set of data decomposition specifications. FORTRAN D is targeted at data-parallel numeric applications that are not supported by existing parallel languages. The extensions proposed in FORTRAN D are compatible with both FORTRAN 77 and FORTRAN 90, a version of FORTRAN with explicit manipulation of high-level array structures. FORTRAN 90D can be viewed as a refinement of CM FORTRAN [TMC89] consistent with a parallel FORTRAN 77.

We believe that FORTRAN D is powerful enough to express most fine-grain parallel computations, but also simple enough that a sophisticated compiler can produce efficient programs for different parallel architectures. In particular, FORTRAN D is well suited for supporting compiler techniques for automatic data decomposition and communication generation, two crucial problems in programming distributed-memory machines. FORTRAN D programs also have the advantage of being deterministic, unlike programs written in most explicitly parallel languages.

We are in the process of implementing a FORTRAN 77D compiler [HKT91b] in the context of the ParaScope programming environment [CCH$^+$88, BKK$^+$89]. We chose as our first target the Intel iPSC/860, a MIMD distributed-memory machine. A later project will produce a FORTRAN 77D compiler for a SIMD distributed-memory machine. We have also started a FORTRAN 90D compiler aimed at the iPSC/860 and NCUBE-2 hypercube [WF91]. We plan to compare how closely FORTRAN D compilers can approach the performance of hand-coded programs, and use our experiences to evaluate the usefulness of FORTRAN D for data-parallel programming.

The rest of this paper presents the design of FORTRAN D, especially its strategy for expressing data parallelism and mapping it to the underlying parallel architecture. We also compare FORTRAN D with related research in parallel languages, data decompositions, and compiler techniques. We start by describing our view of data parallelism.

2 Data Parallelism

The data decomposition problem can be approached by noting that there are two levels of parallelism in data-parallel applications. First, there is the question of how arrays should be *aligned* with respect to one another, both within and across array dimensions. We call this the *problem mapping* induced by the structure of the underlying computation. It represents the minimal requirements for reducing data movement for the program, and is largely independent of any machine considerations. The alignment of arrays in the program depends on the natural fine-grain parallelism defined by individual members of data arrays

Second, there is the question of how arrays should be *distributed* onto the actual parallel machine. We call this the *machine mapping* caused by translating the problem onto the finite resources of the machine. It is dependent on the topology, communication mechanisms, size of local memory, and number of processors in the underlying machine. Data distribution provides opportunities to reduce data movement, but must also maintain load balance. The distribution of arrays in the program depends on the coarse-grain parallelism defined by the physical parallel machine.

FORTRAN D requires the user to specify data decompositions in terms of these two levels of data parallelism. First, the `ALIGN` statement is used to describe a problem mapping. Second, the `DISTRIBUTE` statement is used to map the problem and its associated arrays to the physical machine. We believe that our two phase strategy for specifying data decomposition is natural for the computational scientist, and is also conducive to modular, portable code. Previous projects also include a third intermediate level of parallelism representing a coarse-grain "virtual machine". We do not think this is necessary for our work, although it may be helpful for explicit message-passing programs.

3 Problem Mapping

In FORTRAN D, the `DECOMPOSITION` statement is used to declare a name for each problem mapping. Arrays in the program are mapped to the decomposition with the `ALIGN` statement. The result represents an abstract high level specification of the fine-grain parallelism of a problem. There may be multiple decompositions representing different problem mappings, but an array may be mapped to only one decomposition at a time. All scalars and arrays not mapped to a decomposition are allocated locally.

3.1 DECOMPOSITION Statement

Declaring a Decomposition

The decomposition statement declares the name, dimensionality, and size of a decomposition for later use. A decomposition is simply an abstract problem or index domain. No storage is allocated for a decomposition.

```
DECOMPOSITION A(N)
DECOMPOSITION B(N,N)
```

In this example, A is declared as an one-dimensional decomposition of size N, with elements indexed from 1 to N. B is a two-dimensional N by N decomposition.

3.2 ALIGN Statement

The `ALIGN` statement is used to map arrays with respect to a decomposition. Arrays mapped to the same decomposition are automatically aligned with each other. Alignment can take place either within or between dimensions.

The alignment of arrays to decompositions is specified by placeholders in the subscript expressions of both the array and decomposition. I, J, K, etc... are canonical placeholders indicating the location of dimensions in a decomposition. Array subscripts are fixed; they always consist of the placeholders in alphabetical order beginning with I. The decomposition subscripts can be functions of the placeholders; they specify the alignment of the array with respect to the decomposition.

Exact Match

The simplest alignment occurs when the array is exactly mapped onto the decomposition. In the following example, the arrays X1 and X2 are mapped exactly onto the equivalent dimensions in the decompositions A and B.

```
REAL X1(N), X2(N,N), X3(N,N)
DECOMPOSITION A(N), B(N,N)
ALIGN X1(I) with A(I)
ALIGN X2(I,J) with B(I,J)
ALIGN X3(I,J) with B(I,J)
```

For convenience, placeholders are not required where the mapping is exact. Multiple arrays may also be aligned with the same statement. For instance, the alignments in the previous example could also have been specified with the following syntax.

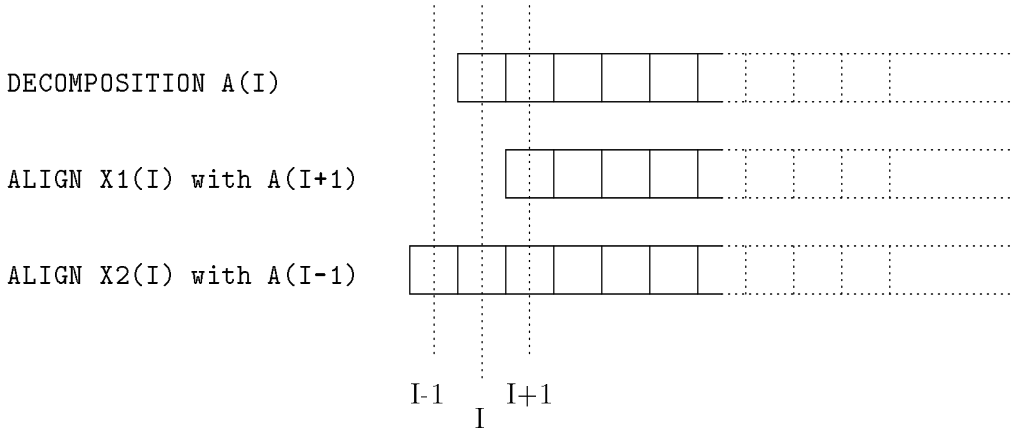

Figure 1 1-D Alignment Offsets

```
REAL X1(N), X2(N,N), X3(N,N)
DECOMPOSITION A(N), B(N,N)
ALIGN X1 with A
ALIGN X2, X3 with B
```

3.2.1 Intra-dimension Alignment

Intra-dimension alignment determines the data decomposition within each dimension. This section describes how offset and stride may be specified.

Alignment Offsets

In Fortran D, the user can specify an alignment offset for any dimension of an array. The simplest case occurs when the array and decomposition have the same number of dimensions. Constants are added to the placeholders in the decomposition to indicate the offset in that dimension.

```
REAL X1(N), X2(N)
DECOMPOSITION A(N)
ALIGN X1(I) with A(I+1)
ALIGN X2(I) with A(I-1)
```

In this example, X1 and X2 are aligned with respect to decomposition A by 1 and -1. X1(I) is thus always mapped to the same element of the decomposition as X2(I+2); *e.g.,* X1(1) is mapped together with X2(3).

```
REAL X3(N,N), X4(N,N)
DECOMPOSITION B(N,N)
ALIGN X3(I,J) with B(I,J-1)
ALIGN X4(I,J) with B(I-1,J+2)
```

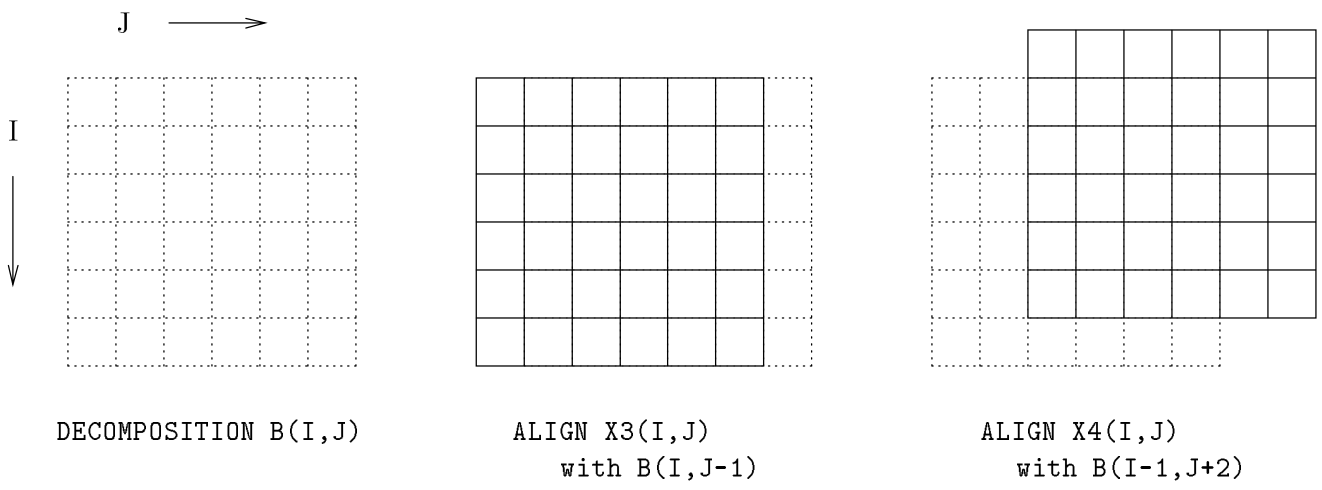

Figure 2 2-D Alignment Offsets

Similarly, in this example the alignment of X3 and X4 with respect to decomposition B means that X3(I,J) is mapped to the same element of B as X4(I+1, J-3).

Alignment Strides

FORTRAN D also allows a stride to be specified when performing intra-dimensional alignment. Alignment strides are used to determine the density of an array mapped to a dimension. They are introduced as coefficients of placeholders in the subscript expressions of decompositions in an `ALIGN` statement. Strides may be also used in combination with offsets.

```
REAL X1(N), X2(N)
DECOMPOSITION A(N)
ALIGN X1(I) with A(2*I)
ALIGN X2(I) with A(2*I-1)
```

In this example, array X1 has a stride of 2 with respect to decomposition A. It is thus mapped to the even elements of A. Array X2 also has a stride of 2, but the alignment offset of -1 causes it to be mapped to every odd element of A. Alignment strides are easily extended to higher order arrays and decompositions, as in the following example.

```
REAL X1(N,N), X2(N,N+N)
DECOMPOSITION B(N,N)
ALIGN X1(I,J) with B(2*I,2*J)
ALIGN X2(I,J) with B(2*I-1,J)
```

Alignment strides with negative values are also allowed; they correspond mapping the reflection of the array dimension onto the decomposition.