The Arc_type Dependence Representation in Polaris

Naraig Manjikian

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada M5S 1A4

May 20, 1995

1 Introduction

In the Polaris compiler [BEF+93], dependences are maintained in `Arc_type` objects, which are a compact form embedded within the nodes representing variable references in the expression trees of a program. Access to dependence information is provided through `DDiterator` objects on `DDGraph` objects. A `DDiterator` object is instantiated with a list of statements and expressions defining the extent of the iteration space. As a `DDiterator` object steps through this space, information in an `Arc_type` object is expanded into an `ActiveArc` object maintained by the `DDGraph`. When an `ActiveArc` is no longer needed, the `DDiterator` permits the `ActiveArc` to be removed, with any modifications to the dependence information reflected into the underlying `Arc_type` object.

It is important to note the following terminology. A dependence *arc* between two expression indicates a possible dependence between them. An arc may have one or more dependence *vectors* associated with it. Hence, both the `Arc_type` representation and the `ActiveArc` representation identify one or more dependence vectors. In Polaris, these vectors are direction vectors.

This document summarizes the format of the `Arc_type` object and how it is manipulated.

2 Data Structures

The `Arc_type` data structure is simply an array of bytes included in objects of type `IDExpr` and `ArrayRefExpr` to serve as a compressed representation of a dependence arc between a pair of expressions. The following type definition appears in `Expression/Expression.h`:

```
typedef unsigned char Arc_type;
```

The following are excerpts from `Expression/IDExpr.h` and `Expression/ArrayRefExpr.h`:

```
class IDExpr : public Expression {
    .
    .
    Arc_type       *_arcs;
    .
    .
};
```

```
class ArrayRefExpr : public Expression {
    .
    .
    Arc_type        *_arcs;
    .
    .
};
```

This _arcs field is used only by *one* of the expressions in a dependence pair, i.e., there is no redundant information. The implication of this is that both a source and target expression must be checked in order to determine whether a dependence exists between the pair of expressions; either one may have the dependence information, but not both. Arc_type does not appear as part of any other data structure.

The storage for Arc_type is allocated in DDgraph::add(). This method is called during the creation of the dependence graph as a DDPairCreator iterator steps through all possible dependence pairs. DDgraph::add() is given dependence information, including direction vectors and source/target expressions, from which it generates the compressed version and places it in the Expression object for the source expression.

When traversing the DDGraph with a DDiterator, ActiveArc objects are used to hold the uncompressed dependence information. A list of ActiveArc objects is maintained by a DDGraph object. Decompression of the information in an Arc_type object is performed by the method DDgraph::initialize_this_active_arc(). This method is invoked by the constructor method for an ActiveArc object when it is created in the method DDgraph::compute_dvf_n_arc(), which is invoked by a DDiterator as it steps through the DDGraph. Source and target expressions are supplied to compute_dvf_n_arc(); both are checked for a valid Arc_type pointer.

The interpretation of the data stored in the array of bytes for Arc_type is embedded in the file ArcSpec.h and the code within DDgraph::add() and DDgraph::initialize_this_active_arc(). A summary of the format for the array of bytes is illustrated in Figure 1. Since the unit of allocation is a byte, there may be unused bits in the last byte. The first 12 bytes are occupied by pointers. The first two pointers are direct copies of Statement* and Expression* pointers. The last pointer is used to maintain a list of Arc_type objects for a given Expression object; the end of this list is marked by NULL.

Following the pointers is a single bit to indicated whether this arc is active or not. When active, a corresponding ActiveArc exists in the DDGraph object as a result of using a DDiterator. This bit is used to avoid creating a duplicate ActiveArc object when more than one DDiterator is used to traverse the DDGraph at the same time. Following the active bit is a sequence of bits describing one or more dependence vectors between the pair of expressions. The start of each vector is indicated by a '1' bit. The end of the list of dependence vectors is marked by a '0' bit, i.e., where the '1' marking the start of the next dependence vector would otherwise appear.

The information maintained for each dependence vector is as follows:

- a bit to indicate the decision (ASSUMED,PROVED)
- a bit indicating the orientation of this dependence vector relative to the pair of expressions; NORMAL indicates that the dependence flows from the source expression to the expression identified by the pointer in first 12 bytes of the Arc_type object; REVERSE indicates that the dependence flows in the opposite direction
- two bits indicating the type of dependence (FLOW, ANTI, INPUT, OUTPUT)

- three bits for each component of the direction vector
- a final three bits which are all zero to mark the end of the dependence vector

References

[BEF$^+$93] B. Blume, R. Eigenmann, K. Faigin, J. Grout, J. Hoeflinger, D. Padua, P. Petersen, B. Pottenger, L. Rauchwerger, P. Tu, and S. Weatherford. Polaris: The Next Generation in Parallelizing Compilers. Tech. Rep. CSRD-1375, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1993.

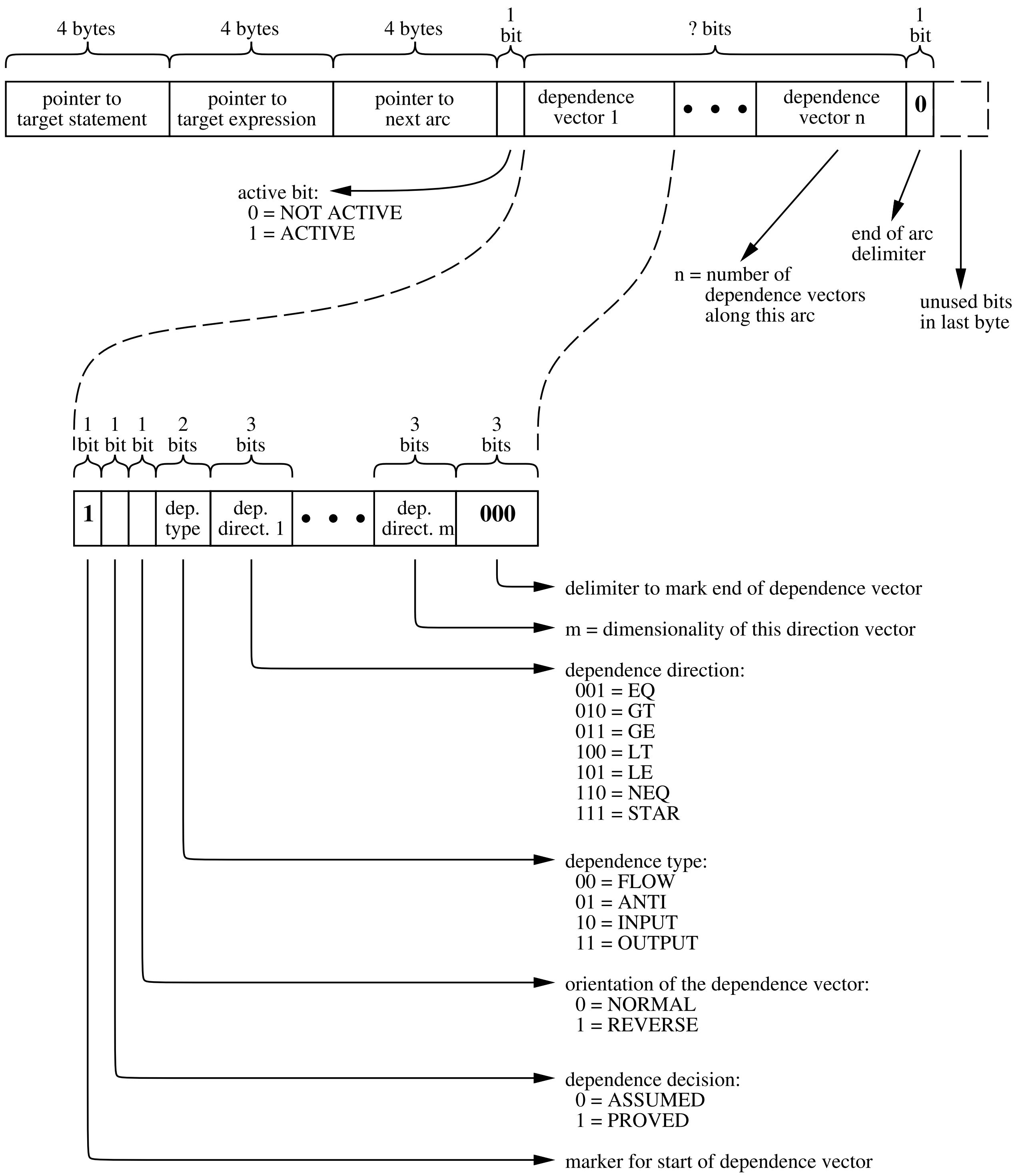

Figure 1: Format for `Arc_type` compressed dependence representation