UMIACS-TR-93-133 December, 1992
CS-TR-3192 Revised April, 1993

Definitions of Dependence Distance

William Pugh

Institute for Advanced Computer Studies
Dept. of Computer Science
Univ. of Maryland, College Park, MD 20742

Abstract

Data dependence distance is widely used to characterize data dependences in advanced optimizing compilers. The standard definition of dependence distance assumes that loops are normalized (have constant lower bounds and a step of 1); there is not a commonly accepted definition for unnormalized loops. We have identified several potential definitions, all of which give the same answer for normalized loops. There are a number of subtleties involved in choosing between these definitions, and no one definition is suitable for all applications.

This work is supported by an NSF PYI grant CCR-9157384 and by a Packard Fellowship.

1 Introduction

Data dependence distance vectors are used to describe loop carried data dependences. For normalized loops, it simply describes the difference in the loop variables between the source and sink of the dependence. The dependence distance can be used to derive information such which loop(s) carries the dependence and whether or not the dependence prevents loop interchange. This paper assumes some familiarity with data dependences and data dependence distances. A good overview is provided by [ZC91].

Let s be an array reference contained in several loops. We use $s(i)$ to refer to the iteration of s when the loop variables of the surrounding loops have the value i (i is a vector). Assume there is a dependence from $s(i)$ to $s'(i')$ (where s' is an array reference, possibly the same as s). Let t be a vector of the loop trip counters corresponding to $s(i)$ (a trip count starts at 1 whenever the loop is started or re-started and is incremented by one in each successive iteration) and let t' correspond to $s'(i')$. We assume there are c common loops around s and s'. Let $step$ be a vector of the step values for the common loops that surround both s and s'.

If $i_{1 \ldots c} = i'_{1 \ldots c}$, the dependence is loop independent. Otherwise, the dependence is carried by the loop $\min\{p | 1 \leq p \leq c \wedge i_p \neq i'_p\}$. We have identified four potential definitions of dependence distance (in the case of normalized loops, all four of these definitions are equivalent):

a The dependence distance is $i'_{1 \ldots c} - i_{1 \ldots c}$.

b The dependence distance is $(i'_{1 \ldots c} - i_{1 \ldots c})/\mathrm{sign}(step_{1 \ldots c})$.

c The dependence distance is $(i'_{1 \ldots c} - i_{1 \ldots c})/step_{1 \ldots c}$.

d The dependence distance is $t'_{1 \ldots c} - t_{1 \ldots c}$. This definition corresponds to normalizing all the loops to have constant lower bounds and a step of 1.

Usually, there is a dependence from several iterations of s to several iterations of s'. In this case, dependence distance from s to s' is described by the union of the dependence distance for each dependent pair. This information can be summarized by the possible signs of the dependence distance or by the range of possible dependence distances.

Figure 1 shows several code fragments, and all of the dependence distances for that fragment, based on each of these definitions.

2 Definition Properties

These definitions each possess certain properties, some of which are listed in Figure 2. Here, we discuss some of these properties in more detail.

#	Code Fragment	Iteration space i	i'	Trip count t	t'	Dependence Distance Def. a	b	c	d
1	`for i1 := 1 to 2 do` `  a[i1]:= ... a[i1-1] ...`	(1)	(2)	(1)	(2)	(1)	(1)	(1)	(1)
2	`for i1 := 0 to 2 by 2 do` `  a[i1] :=  ... a[i1-2]  ...`	(0)	(2)	(1)	(2)	(2)	(2)	(1)	(1)
3	`for i1 := 2 to 0 by -2 do` `  a[i1] :=  ... a[i1+2]  ...`	(2)	(0)	(1)	(2)	(-2)	(2)	(1)	(1)

4	`for i1 := 1 to 2 do`
	`  for i2 := i1 to 2 do`
	`    a[i1, i2] :=  ... a[i1-1, i2]  ...`
		(1,2)	(2,2)	(1,2)	(2,1)	(1,0)	(1,0)	(1,0)	(1,-1)

5	`for i1 := 0 to 1 do`
	`  for i2 := i1 to 3 by 3 do`
	`    a[i1, i2] :=  ... a[i1-1, i2+2] ...`
		(0,3)	(1,1)	(1,2)	(2,1)	(1,-2)	(1,-2)	$(1,\frac{-2}{3})$	(1,-1)

6	`for i1 := 1 to 3 do`
	`  for i2 := max(1, i1-1) to min(2, i1) do`
	`    a[i1, i2] := ... a[i1-1, i2] ...`
		(1,1)
		(2,2)
			(2,1)
			(3,2)
				(1,1)
				(2,2)
					(2,1)
					(3,1)
						(1,0)
						(1,0)
							(1,0)
							(1,0)
								(1,0)
								(1,0)
									(1,0)
									(1,-1)

Figure 1: Dependence distance, according to different definitions, for several examples

Property	a	b	c	d
Dependence distances are always lexicographically positive		$\surd$	$\surd$	$\surd$
Dependence distances are always integral	$\surd$	$\surd$		$\surd$
Directly usable for instruction scheduling			$\surd$	$\surd$
Skewing the loops skews the dependence distances	$\surd$	$\surd$	$\surd$	
Interchanging loops interchanges the dependence distances	$\surd$	$\surd$	$\surd$	

Figure 2: Dependence distance definitions properties

Some transformations, such as loop interchange, depend on the dependence direction (forward, backward, or independent) in each loop. Often, the dependence direction is equated with the sign (positive, negative or zero) of the dependence distance. For normalized loops, as well as definitions b, c and d, this equivalence holds. However, if a is used, the dependence direction must be determined from both the sign of the dependence distance and the sign of the step.

Dependences must be chronological forward: from one array s reference to a later array reference s'. A dependence that is chronological backward must be reversed to represent a dependence from s' to s. For definitions b, c and d, a dependence is chronologically forward if and only if the dependence distance is lexicographically positive. For definition a, the sign of the loop steps needs to be taken into account when determining which dependences are chronologically forward. (Alternatively, whether or not a dependence is chronologically forward can be determined from the direction vector, as described above).

If dependence distances are not guaranteed to be integral, it is more difficult to represent dependence distances in a way that they can be manipulated efficiently and accurately. Alternatively, we could retain the use of integers by marking non-integral distances as unknown (the only known examples that produce non-integral distances using definition c are artificially contrived).

Some loop transformations, such as loop skewing, loop interchange and unimodular transformations [Ban90], are most naturally treated as transformations of the iteration space of the loops. Having information about the dependence in terms of the iteration space makes it easier to determine when these transformations are legal and what their effects are. Using definition d, skewing has no effect on the dependence distance, making it difficult to determine how to skew a loop in order to enable loop interchange (it might be possible to use a combined skew/interchange transformation).

For applications such as instruction scheduling and perhaps synchronization, it is important to know the number of iterations of the loop carrying the dependence between the source and sink of the dependence. This would suggest the use of definition c or d. However, this value can also be calculated from the distance as calculated by definition a or b and the step of the loop that carries the dependence. It is unclear if the trip-count distance for loops inside of the loop carrying the dependence is useful when doing instruction scheduling (this value cannot always be calculated from definitions a, b or c).

3 Potential problems and errors

There are at least two potential traps regarding dependence distance. One is that all parts of a system must communicate consistently with regards to what the dependence distance is. This problem occurred in Parascope: when programs were sent to PFC for analysis, the returned analysis was in terms of definition d, but was interpreted as definition c.

A number of systems, including the KAP and Parafrase-2 compilers (from Kuck and Associates and from Univ. of Illinois) and the Parascope programming environment (from Rice University) all reported that there is no dependence in Example 5. This is apparently because they all attempt to short-cut the GCD test by simply checking if the dependence distance is integral. Example 5 shows that this can lead to false negatives if definition c is used. All three of these systems also found a dependence from `c[2*i]` to `c[2*p+1]` (where `p` had an unknown value), which a correct implementation of the GCD test would have recognized as impossible. We have confirmed that Parascope uses definition c. This bug has been fixed in KAP, and may be fixed in subsequent releases of Parafrase and Parascope.

4 Conclusion

In any system, dependence distance may be used in several contexts. In some of these contexts, definition d might be the appropriate definition, in others, one of the other definitions may be more appropriate. People have been able to avoid deciding whether they need the trip-count distance (d) or some version of the iteration-space distance ($a - c$), since all of these definitions are the same for normalized loops. Plugging any one definition into all existing uses will probably break things. Careful thought needs to be given as to exactly what definition of dependence distance is needed in any application, and several definitions may be needed within a single system.

This unfortunately leaves us with a situation where the term "dependence distance" has several possible meanings. We introduce the new term "dependence difference" for the value calculated by definition a. Within our work, we find this value easy to calculate, has a clean definition, and we can easily change our system to work with dependence direction and difference rather than dependence distance. Definition c might require the fewest changes to existing systems, assuming fractional dependence distances are handled by marking them as unknown distances. However, it may not be the best dependence abstraction on which to build future work.

Whatever choice is made, careful thought needs to be given as to how the system needs to be adapted so as to work with whichever choice is made.

Of course, another approach is to abandon dependence distance all together. A number of researchers have noted that dependence distance and directions are inadequate dependence abstractions for some transformations [Wol91, Pug91, Fea91, MAL93]. However, no alternative has yet gained acceptance.

References

[Ban90] U. Banerjee. Unimodular transformations of double loops. In *Proc. of the 3rd Workshop on Programming Languages and Compilers for Parallel Computing*, pages 192–219, Irvine, CA, August 1990.

[Fea91] Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1), February 1991.

[MAL93] Dror E. Maydan, Saman P. Amarasinghe, and Monica S. Lam. Array data-flow analysis and its use in array privatization. In *ACM '93 Conf. on Principles of Programming Languages*, January 1993.

[Pug91] William Pugh. Uniform techniques for loop optimization. In *1991 International Conference on Supercomputing*, pages 341–352, Cologne, Germany, June 1991.

[Wol91] Michael Wolfe. Experiences with data dependence abstractions. In *Proc. of the 1991 International Conference on Supercomputing*, pages 321–329, June 1991.

[ZC91] Hans Zima and Barbara Chapman. *Supercompilers for Parallel and Vector Computers.* ACM Press, 1991.