Recognizing and Parallelizing Bounded Recurrences
D. Callahan
Tera Computer Company

Abstract This paper examines the problem of recognizing and optimizing a class of recurrences called bounded recurrences which are a generalization of the parallel prefix problem. I show how these recurrences can be executed concurrently and examine the problem of detecting them automatically. The contribution of this paper is a framework for representing information about recurrences and examination of some structural aspects of bounded recurrences. I also examine linear recurrences in some detail showing how to generate efficient code directly from source expressions.

1 Introduction

The historic goal of parallelizing compilers was to recognize so called "implicit" parallelism in sequential programs and to transform those programs so that that parallelism can be exploited by parallel architectures. The core research has been on recognizing when the separate iterations of a FORTRAN DO loop can be executed concurrently and how to transform the loop so that concurrent execution is possible. The primary issue is whether there is an interaction between these iterations which must be respected and hence induces a total or partial order on the execution of the iterations.

A recurrence is a computation of the general form: 2

1 This research sponsored by Defense Advanced Research Projects Agency Information Science and Technology Office ARPA Order No. 6512/2-4; Program Code No. 0T10 issued by DARPA/CMO under Contract MDA972-90-C-0075.

The views and conclusions contained in this document are those of Tera Computer Company and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Unit States Government.

2 Notation: I will use typewriter font for Fortran or similar lower level language, italics and mathematical notation for meta-language constructs and abstractions, and the syntax \([A_1, \ldots, A_n]\) to indicate a vector of values. Generally,
where the values \( A_i \) may be a \( k \)-tuple of separate values. This is a classic example of a loop where the iterations interact and hence does not fit the basic model of a "parallel loop". Recurrences of this form are very common in numerical algorithms and considerable effort has been expended in parallelizing special cases (for examples, see Rodrigue[15]). A quarter of the 24 Livermore Loops[12] are recurrences which can not be parallelized without applying special techniques.

This paper explores a class of recurrences called \textit{bounded recurrences} for which compositions of the functions \( f_i \) can be computed efficiently. This ability allows the above recurrence to be solved using the techniques developed for solving the \textit{parallel prefix} problem. The parallel prefix problem is the special class of recurrences of the form:

\begin{verbatim}
DO I = 1, N
    A_i = A_{i-1} \oplus X_i
ENDDO
\end{verbatim}

where \( \oplus \) is an arbitrary associative binary operator. There are known methods[11] for executing this loop in \( O(\log_2(N)) \) time on an unbounded number processors and \( O(N/P) \) time on \( P \) processors.

The general recurrence case can be parallelized if we are able to compute compositions of the operations \( f_i \). For example, we can rewrite the basic loop into:

\begin{verbatim}
DO I = 1, N
    g_i = g_{i-1} \circ f_i
ENDDO
DO I = 1, N
    A_i = g_i(A_0)
ENDDO
\end{verbatim}

where \( g_0 \) is the identify function. The second loop is trivially parallel and the first is an instance of parallel prefix.\(^3\)

The potential speedup of this transformation depends on the cost of computing compositions and finally applying the composite functions, as well as the number of processors \( p \) applied to the problem. In the simple case of a parallel prefix problem, the cost of composing the functions is equal to the application cost and speedup is limited to \( p/2 \). For more complex recurrences this limit will be lower. The extra work which limits speedup is referred to as \textit{redundant} work.

In the following sections, I will define bounded recurrences and discuss some structural properties which shows how complex bounded recurrences can be recognized based on simpler component recurrences. I will examine in depth the special case of linear recurrences of bounded order giving algorithms to generate code to compute compositions based on input expression trees. I also show how to estimate the amount of redundant work in the parallel code which in turn allows cost-benefit decisions to be made. I will give some examples based on hand-compiled loops running on an instruction level simulator for one Tera processor[4] and discuss related work.

\(^3\)We see that "reductions" are special cases in which only the final \( A_N \) is required and so the second "application" loop can be eliminated.