Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Systems Architecture 50 (2004) 729–742
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

Exploitation of parallelism to nested loops with dependence cycles

Weng-Long Chang [a,*], Chih-Ping Chu [b,1], Michael (Shan-Hui) Ho [a,2]

[a] *Department of Information Management, Southern Taiwan University of Technology, Tainan County 710, Taiwan, ROC*
[b] *Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan City 701, Taiwan, ROC*

Received 24 March 2001; received in revised form 6 June 2004; accepted 10 June 2004
Available online 25 August 2004

Abstract

In this paper, we analyze the recurrences from the breakability of the dependence links formed in general multi-statements in a nested loop. The major findings include: (1) A *sink variable renaming* technique, which can reposition an undesired anti-dependence and/or output-dependence link, is capable of breaking an anti-dependence and/or output-dependence link. (2) For recurrences connected by only true dependences, a *dynamic dependence* concept and the derived technique are powerful in terms of parallelism exploitation. (3) By the employment of global dependence testing, link-breaking strategy, Tarjan's depth-first search algorithm, and a topological sorting, an algorithm for resolving a general multi-statement recurrence in a nested loop is proposed. Experiments with benchmark cited from Vector loops showed that among 134 subroutines tested, 3 had their parallelism exploitation amended by our proposed method. That is, our offered algorithm increased the rate of parallelism exploitation of Vector loops by approximately 2.24%.

Keywords: Parallelizing compilers; Vectorizing compilers; Loop optimization; Data dependence analysis; Dependence cycle; Parallelism exploitation

1. Introduction

In high speed computing [20], there are the two most popular parallel computational models, distributed memory multiprocessors and shared memory multiprocessors. Because the technique to memory hardware [20] is improved, therefore, the access time of shared memory for a system of multiprocessors is obviously decreased and the

* Corresponding author. Tel.: +886 6 6891 421/2533131x 4300; fax: +886-6-2541621.
E-mail addresses: changwl@mail.stut.edu.tw, changwl@csie.ncku.edu.tw (W.-L. Chang), chucp@csie.ncku.edu.tw (C.-P. Chu), mhoincerritos@yahoo.com (M. (Shan-Hui) Ho).
[1] Tel.: +886 6 2757575x62527; fax: +886 6 2747076.
[2] Tel.: +886 6 2533131x4300; fax: +886 6 2541621.

1383-7621/$ - see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.sysarc.2004.06.001

system has been increasingly used for scientific and engineering applications. However, the major shortcoming of shared memory multiprocessors is the difficulty in programming because programmers are responsible for analyzing data dependence relations among statements in programs and exploiting the parallelism of statements in programs among shared memory multiprocessors.

A successful vectored/paralleled compiler is capable of exploiting the parallelism of a program. Three features of a vectored/paralleled compiler determine its level of parallelism exploitation: (1) an accurate data dependence testing, (2) efficient loop optimization and (3) efficient removals of undesired data dependences.

Techniques for dependence analysis algorithms, which directly support data dependence testing, have been developed and used quite successfully [2,5,7–10,20,22–27]. Computationally expensive programs in general spend most of their time in the execution of loops. Extracting parallelism from loops in an ordinary program therefore has a considerable effect on the speed-up. Many loop optimizational methods have been developed and broadly fallen into two classes: loop vectorization and loop parallelization [3,4,20,21,28–30]. In terms of the reduction of data dependences, most researches concentrate on loop optimization by the front-end of vectored/paralleled compilers such *as scalar renaming, scalar expansion*, *scalar forward-substitution* and *dead code elimination* [20]. Studies on the back-end of vectored/paralleled compilers primarily deal with separation of parallelism execution and sequential execution of the statements. Relatively less attention has been given to data dependence elimination [1,9,11].

Recurrence is a type of π-blocks in a general nested loop, which is extracted at the time of loop distribution, a back-end phase [17]. Statement(s) involved in a recurrence are strongly connected via various dependence types. Famous techniques, such as *node splitting, thresholding, cycle shrinking*, etc., are able to eliminate data dependence of a recurrence to a certain extent, depending upon specific dependence types [1,17,18,20,21].

In this paper, we study a parallelism exploitation for loops with dependence cycles on basis of breaking dependence links. Formally the breaking strategies for dependence cycles were surveyed in a single loop [11]. Their method is extended to break the dependence links in a nest of loops. In Section 2, the concept of data dependence is reviewed. In Section 3, an analysis of the formation of dependence cycles is provided and three dependence links on the breaking-strategy basis are derived. For each link pattern, its features, breaking techniques and applications are introduced in detail. An algorithm is developed for the resolution of a general multi-statement recurrence. Experimental results showing the advantages of the proposed method are given in Section 4. Finally, Section 5 contains a conclusion.

2. Data dependence

It is assumed that there are two statements within a general loop. The general loop is presumed to contain n common loops. Statements are postulated to be embedded in n common loops. An array A is supposed to appear simultaneously within statements and if a statement S_2 uses the element of the array A defined first by another statement S_1, then S_2 is true-dependent on S_1. If a statement S_2 defines the element of the array A used first by another statement S_1, then S_2 is anti-dependent on S_1. If a statement S_2 redefines the element of the array A defined first by another statement S_1, then S_2 is output-dependent on S_1. Another dependence, control dependence, which arises due to control statements, is not addressed in this paper.

Each iteration of a general loop is identified by an iteration vector whose elements are the values of the iteration variables for that iteration. For example, the instance of the statement S_1 during iteration $\vec{i} = (i_1, \ldots, i_n)$ is denoted $S_1(\vec{i})$; the instance of the statement S_2 during iteration $\vec{j} = (j_1, \ldots, j_n)$ is denoted $S_2(\vec{j})$. If $(i_1,\ldots,i_n)$ is identical to $(j_1,\ldots,j_n)$ or $(i_1,\ldots,i_n)$ precedes $(j_1,\ldots,j_n)$ lexicographically, then $S_1(\vec{i})$ is said to precede $S_2(\vec{j})$, denoted $S_1(\vec{i}) < S_2(\vec{j})$. Otherwise, $S_2(\vec{j})$ is said to precede $S_1(\vec{i})$, denoted $S_1(\vec{i}) > S_2(\vec{j})$. In the following, Definitions 2.1–2.7, cited from [3,4,11], will be used later.