Collective Loop Fusion for Array Contraction

G. Gao and R. Olsen
McGill University, Montréal, Québec, Canada

V. Sarkar
IBM Palo Alto Scientific Center, Palo Alto, California

R. Thekkath
University of Washington at Seattle

Abstract

In this paper we propose a loop fusion algorithm specifically designed to increase opportunities for array contraction. Array contraction is an optimization that transforms array variables into scalar variables within a loop nest. In contrast to array elements, scalar variables have better cache behavior and can be allocated to registers. In past work we investigated loop interchange and loop reversal as optimizations that increase opportunities for array contraction [13]. This paper extends this work by including the loop fusion optimization. The fusion method discussed in this paper uses the maxflow-mincut algorithm to do loop clustering. Our collective loop fusion algorithm is efficient, and we demonstrate its usefulness for array contraction with a simple example.

1 Introduction

Loop optimization plays a critical role in the compiler optimization of scientific programs. Loops are primarily used to read and write large arrays of values. The cost of array accesses can be reduced by transforming loop nests for cache locality [11, 16, 6]. However, inspite of high cache hit rates, array accesses can incur a large overhead. Because array variables cannot be allocated to registers, load and store instructions must be issued to move the array values between the cache and the register set (these instructions typically comprise about 30% of the total instructions executed). These load and store instructions interfere with instruction parallelism because of serializations imposed by the memory system and by the pessimistic memory aliasing assumptions that must be made by the instruction scheduler. This limitation on
instruction parallelism is a serious problem for modern processor architectures that depend heavily on instruction parallelism to achieve high performance.

Our approach is based upon performing loop fusion to increase the opportunities for array contraction [13]. Array contraction replaces an array variable by a scalar variable or by a buffer containing a small number of scalar variables. This replacement usually eliminates many load/store instructions because scalar variables are good candidates for register allocation [4]. A secondary benefit arises from the fact that loop fusion increases the number of instructions in a loop body.

The algorithm for collective loop fusion presented in this paper is novel in its use of a cluster numbering scheme followed by one or more applications of the maxflow-mincut algorithm [14]. Unlike other loop fusion algorithms that are based upon fusing pairs of loops, our algorithm operates collectively on all loop nests. Our approach is provably optimal for the special case of two-cluster partitioning for which the maxflow algorithm is known to yield a cut set of minimum size. We believe that our approach is generally better than the pairwise greedy approach, though this can only be confirmed by performance comparisons on realistic test cases.

The rest of the paper is organized as follows: Section 2 describes the loop dependence graph which is our program representation. Section 3 formally states the optimization problem, and Section 4 describes our solution. Section 5 presents a case study and experimental results. Section 6 discusses related work, and Section 7 the conclusion and future work.

2 Program Representation

The program representation assumed for performing collective loop fusion is a Loop Dependence Graph (LDG). The loop dependence graph represents a single-entry, single-exit region consisting of \( k \) identically control dependent perfect loop nests [5], with conformable loop bounds. A node in the LDG represents a loop nest [17]; an edge in the LDG represents a loop-independent data dependence from the source loop nest to the destination loop nest [2]. Each edge in the LDG is marked as being fusible or nonfusible. The source and destination loop nests of a nonfusible LDG edge cannot be fused because this would violate the data dependence test for loop fusion [17]. Fusible edges are further classified as contractable and noncontractable. A contractable edge is one where loop fusion will yield a savings in memory cost because a data computed by a source loop nest will be available in the processor’s registers for use by a destination loop nest. A noncontractable edge is one for which loop fusion will not yield such a savings in memory cost. Only flow edges can be marked contractable. Figure 1 contains a simple example with six FORTRAN loop nests. The Loop Dependence Graph for Figure 1 is shown in Figure 2(a) in Section 4, where the nonfusible edges are marked with \( X \) and the fusible-noncontractable edge