Chapter 31

GENERALIZED DATA TRANSFORMATIONS

V. Delaluz¹, I. Kadayif¹, M. Kandemir¹ and U. Sezer²
¹ CSE Department, The Pennsylvania State University, University Park, PA 16802, USA;
² ECE Department, University of Wisconsin, Madison, WI 53706, USA;
E-mail: {delaluzp,kadayif,kandemir}@cse.psu.edu, sezer@ece.wisc.edu

Abstract. We present a compiler-based data transformation strategy, called the “generalized
data transformations,” for reducing inter-array conflict misses in embedded applications. We
present the theory behind the generalized data transformations and discuss how they can be inte-
grated with compiler-based loop transformations. Our experimental results demonstrate that the
generalized data transformations are very effective in improving data cache behavior of
embedded applications.

Key words: embedded applications, data transformations, cache locality

1. INTRODUCTION

In many array-intensive embedded applications, conflict misses can consti-
tute a significant portion of total data cache misses. To illustrate this, we
give in Figure 31-1, for an 8 KB direct-mapped data cache, the breakdown
of cache misses into conflict misses and other (capacity plus cold) misses for
seven embedded applications from image and video processing.¹ We see that,
on the average, conflict misses consist of 42.2% of total cache misses. In fact,
in two applications (Vcap and Face), conflict misses constitute more than 50%

![Figure 31-1. Contribution of conflict misses and other (capacity plus cold) misses.](image-url)

of all misses. The reason for this behavior is the repetitive characteristic of conflict misses; that is, they tend to repeat themselves at regular (typically short) intervals as we execute loop iterations. The small associativities of the data caches employed in embedded systems also contribute to large number of conflict misses.

A well-known technique for reducing conflict misses is array padding [6]. This technique reduces conflict misses by affecting the memory addresses of the arrays declared in the application. It has two major forms: intra-array padding and inter-array padding. In intra-array padding, the array space is augmented with a few extra columns and/or rows to prevent the different columns (or rows) of the array from conflicting with each other in the data cache. For example, an array declaration such as $A(N, M)$ is modified to $A(N, M + k)$ where $k$ is a small constant. This can help to prevent conflicts between two elements on different rows. Inter-array padding, on the other hand, inserts dummy array declarations between two original consecutive array declarations to prevent the potential conflict misses between these two arrays. For instance, a declaration sequence such as $A(N, M), B(N, M)$ is transformed to $A(N, M), D(k), B(N, M)$, where $D$ is a dummy array with $k$ elements. This affects the array base addresses and can be used for reducing inter-array conflicts. While array-padding has been shown to be effective in reducing conflict misses in scientific applications [6], there is an important factor that needs to be accounted for when applying it in embedded environments: increase in data space size. Since data space demand of applications is the main factor that determines the capacity and cost of data memory configuration in embedded designs, an increase in data space may not be tolerable. To evaluate the impact of array padding quantitatively, we performed a set of experiments with our benchmark codes. Figure 31-2 gives the percentage reduction in (simulated) execution time (for an embedded MIPS processor with an 8 KB data cache) and the percentage increase in data space when array

![Figure 31-2. Impact of array padding.](image)