

Compiler Transformations for Improving the Utilization of Hardware Prefetching Units

Ibrahim Hur and Calvin Lin
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

In collaboration with Brian O’Krafka
IBM Enterprise Systems Group
Austin, TX

1 Abstract

The increasing use of sophisticated latency-hiding mechanisms in microprocessors presents new challenges and tradeoffs for compilers. This project will study such tradeoffs for modern microprocessors, as exemplified by the POWER3. We will initially focus on the interaction between hardware prefetching and locality of reference. In particular, we will develop a cost model that captures the effects of hardware prefetching, we will experimentally evaluate our model using detailed simulation, and we will use our model to guide new compilation strategies.

2 The Problem

The difficulty of compiling for modern microprocessors is illustrated by considering the tradeoff between prefetching and locality. For example, loop fusion can improve locality of reference by enabling array contraction, which collapses a temporary array to a scalar. However, for a machine like the POWER3, we may instead wish to increase the number of data streams that can be prefetched, thereby feeding data to the functional units in a regular manner that supports pipelined execution. To accomplish this, we would perform loop fission and store values in temporary arrays that can be accessed as a stream, but this increases the amount of data used and can pollute the cache. We hope to devise methods of deciding between these two competing transformations.

3 Proposed Work

We will explore and compare various techniques for modulating the number of threads:

- Loop fusion creates larger loop bodies, which can increase the number of streams by increasing the number of arrays accessed in a loop nest.

- Loop fission creates smaller loop bodies, which can reduce the number of streams by reducing the number of arrays accessed in a loop nest.
- A combination of loop splitting and loop fusion breaks each stream into multiple smaller streams. For example, the following loop

```
for i=1 to 100
  a[i] = b[i] + c[i];
```

could be transformed to the following loop to double the number of active streams:

```
for i=1 to 50
  a[i] = b[i] + c[i];
  a[i+50] = b[i+50] + c[i+50];
```

- Interleaving the storage of arrays can combine multiple disjoint streams into a single larger stream.
- Statements can be reordered and restructured to produce more regular access patterns.

In conjunction with these transformations, we will investigate policies for inserting instructions that kick-start the hardware prefetch mechanism. This should make prefetching viable for shorter streams.

All of these techniques have effects on locality, and we will study these effects, too. For example, we will investigate new policies for tiling and padding, which may have different optimal tile sizes in the presence of hardware prefetching mechanisms. We will also consider other data restructuring techniques, such as improving locality and prefetching potential by creating temporary copies of physically dispersed data. Finally, we expect that future architectures will expose additional features to the software, such as mechanisms for bypassing the cache, and we eventually would like to study these features, as well.

4 Experimental Evaluation

We will evaluate our work on at least two sequential platforms: an out of order, speculative execution microprocessor simulator and a 630-based machine. We will also evaluate our work on the Cray T3E, where the hardware prefetching mechanism can be controlled by software.

Although data prefetching is generally considered useful for scientific applications, we will also investigate its effects on commercial workloads. In this vein, we hope to examine DB2 source code for possible compiler optimizations that can make use of hardware prefetching.