Speculative Parallelization in Decoupled Look-ahead Architectures

Alok Garg, Raj Parihar, and Michael C. Huang
Dept. of Electrical & Computer Engineering
University of Rochester
Rochester, NY 14627, USA
Email: {garg, parihar, mihuang}@ece.rochester.edu

I. INTRODUCTION AND MOTIVATION

One well known approach to mitigate the impact of branch mispredictions and cache misses is to enable deep look-ahead so as to overlap instruction and data supply with instruction processing. A continuous look-ahead process which uses separate thread of control on another hardware contexts is one such approach which we call decoupled look-ahead [1], [2]. However, in such look-ahead schemes, look-ahead thread can often become the performance bottleneck. In this work, we explore speculative parallelization in a decoupled look-ahead agent. Intuitively, speculative parallelization is aptly suited to the task of speeding up look-ahead agent for two reasons. First, the program slice for look-ahead does not contain all the data dependencies embedded in the original program, providing more opportunities for parallelization. Second, the execution of the slice is only for look-ahead purposes and thus the environment is inherently more tolerant of dependence violations.

II. ARCHITECTURAL OVERVIEW

A. Software Support

To detect coarse-grain parallelism suitable for thread-level exploitation, we use a profile guided analysis tool. The look-ahead thread binary is profiled to identify dependencies and their distances. Two distantly dependent basic blocks can be potential candidate for parallel execution. With the information from profile-based analysis, we find static code locations to spawn off parallel threads, and locations where the new threads can start their execution.

B. Hardware Support

Hardware support needed for speculative parallelization is very similar to the basic support needed for multi-threading. However, look-ahead is not correctness-critical which provides enormous flexibility in choosing what to support and what can be simply skipped in conventional multi-threading environment. In our system, spawned thread executes a future code segment in the same logical thread. If everything is successful (no dependencies violated), it is the primary look-ahead thread which did the spawning will terminate. When a new thread is spawned, it inherits the architectural state including memory content and register state and the starting PC of the new thread is also recorded.

C. Runtime Spawning Control

At runtime, two thread contexts are reserved for look-ahead. A spawn instruction is handled at dispatch time and will freeze the pipeline front end until the rename table is duplicated and a new context is set up. If another thread is already occupying the context, the spawn instruction is discarded else gets the opportunity to spawn.

III. EVALUATIONS AND RESULTS

Our speculative parallelization strategy provides up to 1.39x speedup over baseline look-ahead (w/o speculative parallelization) [2]. On average, measured against the baseline look-ahead system, the contribution of speculative parallelization is a speedup of 1.13x as shown in figure 1.

![Figure 1. Speedup of baseline look-ahead and speculatively parallel look-ahead over single-core baseline.](image)

IV. CONCLUSION

We proposed a hardware/software mechanism to apply speculative parallelization to look-ahead thread. To best of our knowledge, this is the first attempt to apply speculative parallelization for efficient look-ahead. This approach works because of two reasons: 1. Look-ahead code contains fewer dependencies thus lends itself to (speculative) parallelization; and 2. Without correctness constraints, hardware support for the look-ahead thread can be much simpler.

REFERENCES
