Using Combined Profiling to Decide When Thread Level Speculation is Profitable

Arnamoy Bhattacharyya
University of Alberta
Edmonton, Alberta, Canada
arnamoy@ualberta.ca

ABSTRACT
Thread Level Speculation (TLS) speculatively executes parts of a program in parallel. Statically determined may dependences between store-load pairs prevent the compiler from speculatively executing parts of programs (e.g. loop iterations or functions). If a compiler can determine that the probability of a may dependence occurring at runtime is low, then it can use TLS to execute the loop in parallel. This research will develop a may dependence profiling framework that is able to capture the effect of different inputs on the dependence behaviour of the program during runtime, using a technique called Combined Profiling (CP) [1]. The dependence profiling will be made efficient using the output from static analysis. TLS code generation strategies will be implemented in a version of the LLVM compiler that will generate code for the hardware support for TLS in the IBM BlueGene/Q (BG/Q) machine.

Categories and Subject Descriptors
D.1.3 [Concurrent Programming]: Parallel Programming

General Terms
Performance

Keywords
Thread Level Speculation, Loop Chunk

1. RESEARCH PROBLEM
(1) Given the BG/Q hardware support for TLS, could a dynamic analysis based on information gleaned from multiple runs of a program make better decisions about when to speculate than a strictly static analysis? (2) How can static dependence analysis be used to make dependence profiling more efficient and more effective in its ability to inform the compiler speculation decisions?

2. MOTIVATION AND BACKGROUND
For each store-load pair in a loop, if all are independent, the loop is parallel, if any pair is dependent, the loop is sequential. Speculation is concerned with may dependences which may or may not materialize at runtime. To decide when to speculate a loop with may dependences, a compiler has to weigh the benefit of a successful parallel execution against the overhead of thread creation and miss speculation. This decision can be based only on a static analysis, or it can also use information obtained through runtime profiling. Multiple runs of the program with different inputs should provide more precise information to estimate the probability of dependence materialization. Berube has proposed a technique called Combined Profiling (CP) to combine the information from multiple profiles for different inputs. This research will adapt CP to profile may dependences.

In the BG/Q the speculative state is stored in the L2 cache and the multiple speculative threads compete for this space in the same way that multiple threads compete for cache storage space in a non-speculative architecture. Therefore a more sophisticated analysis that takes into consideration the cache organization and the offsets of the memory references may be required to yield a better estimation of the profitability of speculating.

3. TECHNIQUES AND RESULTS
After the identification of speculation candidates, the proposed profiling framework resembles loop tiling. It first determines the memory footprint of a loop iteration. If the footprint size is greater than the speculative buffer size, the loop is not speculated. Otherwise the framework determines the number of iterations, a chunk of the loop, that can fit into one thread. For each chunk the framework profiles memory accesses by the may-dependent store-load pairs and records the probability of the dependence occurring at runtime. By querying a histogram built with the probabilities for multiple inputs, the compiler makes speculation decisions.

A preliminary examination of the SPEC CPU 2006 benchmark suite using the existing loop-dependence analysis pass in LLVM reveals that out of approximately 3 million store-load pairs that appear in loops, 46.7% are independent and 53.24% are regarded as may-dependent. This indicates that there is significant scope for the application of TLS to this set of benchmarks.

4. REFERENCES