Summary Part 1
Data Dependence Profiling for Speculative Optimizations

Mrinal Nath (ID: 3307043)
February 21, 2005

What are the problems solved by this paper?

Traditional data dependence analysis algorithms are not good enough for programs written in C/C++ due to the use of pointers in these languages. Alias analysis provides less accurate information compared to data dependence analysis. Hardware support for data dependence may not always be available. Even if it is available, the hardware cost limits the window size available to the hardware for speculation. This can be a severe limitation.

Also, since the cost of mis-speculation can be high, speculative optimizations must be done carefully. Even complicated static analysis of programs does not provide enough information to the compiler for it to make well-educated optimizations. Thus, it is useful to have detailed information regarding data dependences while doing such optimizations.

However, software based data dependence profiling can be very time consuming. This paper proposes techniques by which an efficient software tool for data dependence profiling can be built. The proposed techniques are able to produce detailed data dependence information for nested loops and summarized dependence information for procedure calls. The techniques also provide distance-vectors for loops and dependence probability information to the compiler. Also, various methods are proposed to reduce the overheads associated with such detailed data dependence profiling.

What are the approaches attempted by this paper?

An instrumented program is used to collect information about the memory references in a program. Pair-wise address comparison to detect data dependences is very expensive. So a shadow memory is used to efficiently detect the data dependences. All four types of dependences are recorded. To correctly profile data dependences between function calls, the calling context of each memory reference is recorded. This is to avoid false dependences between function calls. In case of loops, it is useful to know the dependence distance. The loops are instrumented to have iteration vectors, and dependence distance information is collected using these iteration vectors. A data dependence between two memory references may occur rarely, frequently or always. If this dependence probability is known, then the compiler can speculate only on the rarely occurring dependences, so that mis-speculations are minimized.

Collecting all this information can be very time and space consuming. So some methods are proposed to reduce the time and space costs. The instrumentation point is selected after certain compiler optimization passes (after register promotion), so that the total number of memory references are reduced. The shadow memory size can be minimized by reducing the granularity (or byte resolution) and also by reducing the size of the iteration counter in the shadow. Sampling techniques can be used to further reduce the amount of memory references tracked. However, since a data dependence is a correlation between two memory references, we have to ensure that both the references are sampled at any given point. This can be ensured if we can sample an entire program segment. So, along with the instrumented version of the program, an uninstrumented version is maintained. During execution,
we switch between these two versions at a predetermined sampling rate, so that complete segments of programs are sampled.

What are the main conclusions of this paper?

The basic scheme of using shadow memory is simple, and allows for various extensions that can be used to gather useful data. The profile information collected is very useful to the compiler for performing speculative optimizations and speculative thread generation. The data dependence analysis provides more accurate information compared to alias analysis techniques. Aggressive code scheduling can be done if good data dependence profile is available to the compiler. The techniques presented in the paper are insensitive to the actual input set used to generate the profile.