

Summary Part 2

Data Dependence Profiling for Speculative Optimizations

Mrinal Nath (ID: 3307043)

February 24, 2005

Did this paper address an important issue? Explain.

Data dependence analysis is used in several compiler optimizations and loop parallelization techniques. It also provides better accuracy about data dependences than alias analysis based techniques. However, general techniques for data dependence analysis are inadequate when pointer-based expressions are used like in C/C++. Also, the cost of mis-speculation can be very high, and so the compiler has to be conservative while Speculatively reordering code. Some architectures provide hardware support for speculation, but the amount of support that can be provided in hardware is inherently limited due to the limited window sizes and cost of hardware.

All these problems limit the amount of information that can be used for data dependence analysis. Highly speculative microprocessors, VLIWs and multithreaded processors depend heavily on the compilers ability to schedule code so that they can achieve maximum performance. Due to the above mentioned limitations, the compiler can not fully exploit the potential of data speculation based code reordering.

Hence, to provide the compiler with detailed data dependence information, profiling techniques are used. This paper describes techniques that can be used to gather highly useful information (e.g. dependence distance, probabilities, etc.) regarding the data dependences in the program while minimizing the overheads associated with the profiling process. Hence, the paper is dealing with an important issue.

Are the proposed approaches valid? Describe its strength and weakness.

Several approaches are proposed to gather a wide variety of profile information regarding the data dependences in the program. Some techniques are presented to reduce the overhead associated with the profiling process. All these techniques are valid.

Strengths

Shadow memory is used to detect data dependences, which is a very efficient technique since it avoids pair-wise address comparison between memory references to detect data dependences. The techniques presented can track data dependences between functions and procedures. Detailed information can be obtained regarding important dependence parameters like dependence distance and dependence probability. Dependence distance information can be used for efficiently parallelizing loops, and dependence probability information can be used to avoid moving code beyond high-probability dependences so that mis-speculations are minimized. Another important strength is that the profiling data is quite accurate and insensitive to input set variations. The different methods to reduce the overheads provide very significant reduction in the overhead, without compromising the quality of the results much.

Weaknesses

There aren't any weaknesses as such. The whole process of gathering profile information takes some overhead (which I guess can't be avoided, given that such detailed information is being collected).

Another thing to be considered is the ease of instrumentation of the program to gather the profile data.

Do the results support the conclusions? Explain.

The authors compared the performance of their scheme against alias-based profiling. For PRE, the results show that programs optimized after data dependence profiling performed much better than programs optimized after alias profiling. The runtime of the programs were reduced, and also the number of retired loads was reduced. For code scheduling optimizations, the results show that a large number of loads could be speculated with a high degree of accuracy. Thus, the results demonstrate the usefulness of data dependence profiling in guiding compiler optimizations for speculative execution.

Describe the potential future works?

As of now, some of the benchmark programs are not handled by the implementation. So the authors can work towards generalizing their techniques so that all types of programs can be profiled (and eventually benefit from compiler optimizations). Also, the technique can be extended to include control-flow profiling. Then with a single profiling run, information about the control-flow profile and the data-flow profile can be gathered and fed back to the compiler for all types of optimizations.