Managing Control and Data Dependence in Future Out-of-order Microarchitectures

D. Morano, D.R. Kaeli
Northeastern University
dmorano, kaeli@ece.neu.edu

A.K. Uht
University of Rhode Island
uht@ele.uri.edu

11th December 2002

Current high performance superscalar processors have widely employed instruction level parallelism (ILP) to achieve their high single-threaded performance levels. The primary mechanisms used in these microarchitectures for implementing their ILP extraction have been the issue window, multiple function units, reorder buffer, and the architected register file. Instruction windows can presently handle about 128 instructions and larger instruction windows are still likely. Current microarchitectures also widely use speculative instruction execution through the prediction of future control flow. This speculative execution is primarily done by predicting the outcomes of individual condition branches and then speculatively executing instructions on the predicted control flow path. Although successive conditional branches can be and are also predicted, speculative execution essentially only proceeds further on a single speculative control-flow path. Proposals have been made to execute along more than one speculative control-flow path simultaneously (multipath execution) but several proposed techniques for doing this incur high additional cost in control logic. Either additional entire reorder buffers are introduced for the alternative speculative paths or some other additional specialized control logic needs to be introduced. As a result, multipath execution has not as yet been considered to be cost effective to do in present processors. There have also been proposals to predict data values for program variables. These include not just branch target addresses or memory operand addresses, but also register and memory operand data values. Although this approach seems promising, it adds a substantial additional logic burden to the control portion of any microarchitecture that employs the technique. Finally, there are proposals to facilitate the execution of control-independent instructions beyond unresolved conditional branches. This technique is also generally expensive in terms of control logic needed to properly coordinate.

Another complication for existing and future processors is the large integrated circuit routing congestion for access to their centralized resources. These centralized resources generally include the issue window, the architected register file, the reorder buffer, and possibly the load-store queue for memory operands. Further, as the number of instructions in flight increases, the need for access to the centralized resources of the microarchitecture increases proportionally and contention for limited ports (on any of the centralized resources named above) begins to impose diminishing performance returns through the forced waiting for resource read or write port access. As a results of these problems, present processors are rapidly nearing the limits of what is possible with the exploitation of ILP through larger instruction windows.

Because of many of the present and foreseeable problems associated with elaborate control logic for dynamically scheduled superscalar processors, alternatives such as explicit parallel instructions architectures or multithreaded microarchitectures are being aggressively explored. However, neither of these approaches has yet been proven to substantially increase execution performance of a single thread of control, which is still an important goal for high performance computing. In spite of the present problems associated with instruction and operand dependence management for dynamically scheduled processors, it is still advantageous and desirable to explore ways to combine many of the promising microarchitectural techniques mentioned already, such as data value prediction,and speculative multipath execution, and execution of control-independent instructions. We present a new approach for the simultaneous management of speculative instruction execution, speculative multipath execution, and for the ordering of speculative and non-speculative operands. Our approach both unifies the handling of control-flow dependencies and data dependencies as well as reducing the amount of control logic needed for large or very large instruction window sizes.

Our approach to program dependency ordering (both control-flow and data-flow) is realized by tagging all instructions that enter into consideration for execution (non-speculative or otherwise) with tags that reflect the dynamic program order. These tags essentially store a time stamp of a sort to place the instruction into its proper dynamic order as compared with all other instructions being executed. We term these tags *time-tags* in light of the way that they are used. We also dynamically predicate all instructions, within the microarchitecture itself, to allow for large amounts of possibly random out-of-order speculative execution. This feature allows for the benefits of predicated execution to be applied to those processor architectures that do not have architecturally visible predication. This technique is still applied even for processor instruction sets that do have predicates also. This proposal is thus independent of any compiler changes and allows for execution of all legacy program codes without recompilation. We further tag all program operands, whether they be register operands, memory operands, or the dynamically introduced microarchitectural predicate operands that are now associated with all instructions. For those architectures that may already have architected predicate registers, these are just registers as far as our technique is concerned and are simply treated as such without loss of any generality. Our use of dynamic predication also allows for an easy way (low hardware cost) to govern the execution of control-independent instructions.

Time-tags are just small integers that have a number of bits that is approximately log-base-two the number of instructions allowed in flight in any given machine implementation. The value of a time tag is incremented for each successive instruction that enters into execution consideration. Generally, that instruction that is oldest in program ordered time and nearest to program commitment is assigned a time-tag value at or near zero, while those instructions (both non-speculative and speculative) that are younger and more speculative are assigned successively higher values. These tags are periodically uniformly decremented throughout the whole of the execution core in order to prevent overflow related issues. In our scheme, all instructions that have entered into consideration for execution remain in a microarchitectural structure that resembles an issue slot or a reservation station until the instruction is retired (either squashed or committed).

To allow for value prediction, instructions are allowed to issue to function units repeatedly as needed or desired until they are retired. Upon instruction issue, although a function unit is allocated for execution, the instruction also stays in its assigned reservation station. To facilitate multipath execution, a path identifier is also assigned to each instruction that enters into execution consideration. This is needed since more than one instruction can have the same relative position in program ordered time (as compared with the last committed instruction) but still have to be disambiguated since they reside on different speculative paths of control. The combination of the architected name for an operand (either its architected register address or its memory address) together with the path identifier and the time-tag form a unique microarchitectural name for the operand. Thus the microarchitecture essentially provides for the full renaming of all architected operands.

Once all instructions and operands are tagged using our technique, all instruction are allowed to issue and execute in any order and with any input operands regardless of whether the input operands are correct or not. In those circumstances when an input operand is not the final committed program value, the instruction will dynamically figure out through operand snooping that it will need to re-execute at least one additional time (with the correct committed input operand value) before it can be committed (if it is even going to be committed). Instructions that are both on the eventual committed program path as well as those that are on alternative speculative paths are allowed to execute simultaneously. Input operands to instructions later in program-ordered time snoop for output operands from previous instructions. Executions or re-execution only need occur when input operands change from previous values. Eventually, all correct operands are snarfed and if the oldest dispatched instructions have executed at least once, commitment of those instructions can occur. This amount of flexibility allows for increased levels of instruction parallelism since more parallelism is being exposed from the program itself.

We present the simulated results of a representative microarchicture using the ordering technique discussed. SpecINT benchmarks were compiled using a standard vendor compiler and then run on the simulator using a current representative memory hierarchy and memory access latencies. Instructions per clock (IPCs) numbers for each benchmark is provided for varying machine configurations and range from about 3.8 to 5.2.