General Purpose VLIW Processors using Replay Buffers

Kevin W. Rudd
Stanford University

Premise: By relying exclusively on compiler-based scheduling and resource management, VLIW processors offer an efficient and scalable platform for exploiting instruction-level parallelism across a wide range of applications and system configurations.

Problem: Because there is no dynamic analysis used in VLIW processors, these processors are intolerant of unexpected execution behavior. Efficiently managing Dynamic Events—interruptions (interrupts, exceptions, and thread switches) and delays (late results)—is required both to ensure compatibility across configurations as well as to allow the use of variable latency features such as data caches. VLIW processors have not been able to manage Dynamic Events efficiently and thus have not been effective in general purpose computing environments.

Previous solutions: Traditional solutions to manage Dynamic Events in VLIW processors result in poor overall performance and high per-event penalties: a combination of restrictive scheduling and special-case hardware has been used to manage interrupting events; global stalls have been used to manage delaying events. Neither technique is efficient or scalable: restrictive scheduling penalizes overall performance; special case hardware and global stall signals complicate designs and can increase the clock cycle time as well as limit scalability.

Our solution: By placing a FIFO-like structure, the Replay Buffer, between the result source (typically a pipeline) and sink (often a register file), all Dynamic Events are trivially managed. When necessary, Replay Buffers collect and replay results to preserve pipeline state across an event. The depth of a Replay Buffer is the same as the exposed depth of the pipeline. A simple configuration consisting of a single replay buffer is shown in figure 1. Replay Buffer (and result selection multiplexor) control is computed using the current thread identifier (the only global signal required), the thread identifier from the result, and the contents of the replay buffer; because of this simple control, Replay Buffers are readily scalable to support multiple threads and wide processor widths.

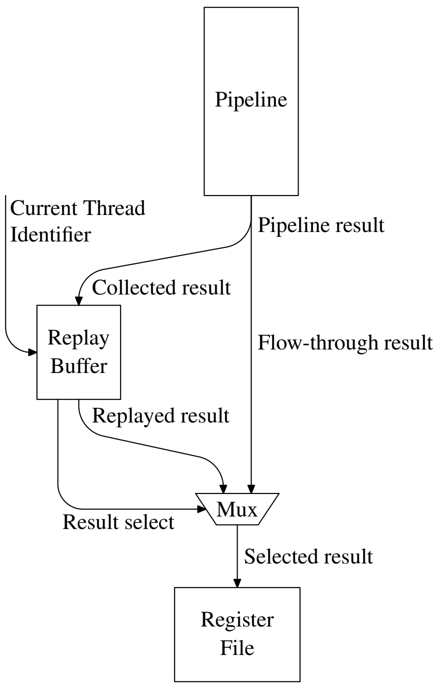

Figure 1: Pipeline to register-file datapath using a single Replay Buffer to manage Dynamic Events.

Dynamic Events are managed by treating each event as a thread switch; Replay Buffers automatically preserve pipeline state from non-current threads and provide zero-cycle thread switches. Three different classes of threads are supported: normal threads (each requiring a Replay Buffer), T_i, that support all Dynamic Events and are used for most user and system execution; an interruption thread, T_X, that supports no Dynamic Events and is used only for low-level processor control during interrupting events; and a delaying thread, T_D, that is used during delaying events and implicitly executes no-operations until the late event is resolved. As long as no context switch is required (such as when using lightweight threads or some system calls) no program (user or operating system) need be aware of the presence of Replay Buffers; context switches do require operating system awareness and may require Replay Buffer contents to be saved and restored.

Conclusion: Using Replay Buffers, VLIW processors overcome their main liability—the inability to manage Dynamic Events efficiently—without resorting to costly hardware or software techniques. Replay Buffers also enhance VLIW processors by providing a general facility for zero-cycle thread switches to both user and system programs. The addition of Replay Buffers to VLIW processors finally makes efficient and scalable instruction-level parallelism available to general purpose computing environments.