Wild and Crazy Ideas held in conjunction with 16th International Conference on Architectural Support for Programming Languages and Operating Systems. March, 2009. Washington DC.

EXtreme Virtual Pipelining (XVP): Moving Towards Scalable Multithreaded Processors

Korey Sewell*, Trevor Mudge*, Steven K. Reinhardt*†

*University of Michigan – Ann Arbor
{ksewell, tnm}@eecs.umich.edu

†Advanced Micro Devices (AMD)
steve.reinhardt@amd.com

1. INTRODUCTION

The impact of simultaneous multithreading (SMT) on small-scale designs (2-4 threads) has been successfully demonstrated in both academia and industry, but large-scale SMT (16-32 threads) is still impractical due to the adverse side-effects that both increasing and sharing common resources would present on such architectures. Achieving efficient, large-scale SMT processors would not only enable higher-throughput multicore chips, but also open up new research angles into parallel architectures and their applications (e.g. speculative multithreading, fine-grained synchronization and communication, virtual machines, etc.).

A major scalability issue in conventional SMTs is the requirement to replicate per-thread structures and to increase the size of shared resources as the number of threads increase. These SMT stipulations are costly because large, critical path components directly affect processor clock rate.

The ability to efficiently share resources amongst hardware threads is also a significant issue for SMT scalability. Current resource distribution techniques alter CPU behavior based on monitoring and reacting to CPU performance counters (e.g. per-thread resource usage). While these methods can be effective on a small-scale, the task of finding optimal resource distributions for threads of various workload behaviors dramatically increases in complexity when considered for a large-scale SMT machine.

Instead of architecting a large-scale SMT through extensive increases in CPU resources and distributing those resources via complex analysis of workload behavior, we propose a solution called e*Xtreme Virtual Pipelining (XVP)* which seeks to virtualize all stalling processor resources. At each stage in the pipeline, XVP provides each concurrent thread the illusion that it has all the processor resources to itself. XVP's virtualization also gives resources the opportunity to configure themselves on-demand rather than attempt to achieve optimal sharing through indirect mechanisms. Since XVP essentially provides a CPU with larger processor components than normal, we also theorize that XVP can facilitate speedup in single-thread processors.

2. EXTREME VIRTUAL PIPELINING

In *eXtreme Virtual Pipelining* (XVP), resources that can stall a pipeline when full are virtualized by memory-mapping their entries into a thread's address space similar to Oehmke's logical register virtualization in VCA [1]. If a CPU has 64 instruction queue (IQ) entries then XVP would map space for 64 IQ entries in each thread's address space. This presents the virtualized view of a full allotment of pipeline resources for each hardware thread and allows threads to withstand "resource full" conditions that would stall a traditional SMT. Figure 1 shows a simplified view of how resources will be virtualized. XVP proposes to virtualize the fetch buffer (FB), IQ, load/store queue (LSQ), register file (RF) and reorder buffer (ROB).

Each "stallable" component in a XVP pipeline stage is matched with a Fill-Spill-Unit (FSU) responsible for inserting and removing entries on an on-demand basis. Heavily contended resource buffers will now serve as a "cache" of a larger resource space giving us the important benefit of resources that dynamically partition themselves for arbitrary workload mixes. For example, instructions dependent on a load miss will be spilled out of the IQ and filled only after the miss is resolved. We can also keep only the most recently used logical registers in the physical register file, while rarely used registers can be spilled to memory. Alternative designs for XVP might also leverage some type of prediction engine or add extra pipeline stages to facilitate fill-spill functionality.

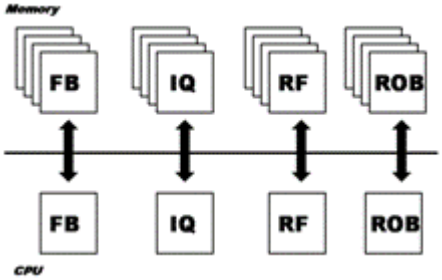

Figure 1 – XVP Virtualizes Pipeline Resources to Memory

Because XVP effectively extends the notion of a hardware context to include pipeline resources, XVP adds a separate "Context" L1 Cache (C-Cache) for context data. Consequently, XVP avoids Data(D)-Cache thrashing by recognizing that context data will have different temporal locality then instruction data. Since register data is spilled to the D-Cache in conventional designs, the C-Cache's decoupling of context data from instruction data has the potential to reduce memory footprint size and the subsequent amount of space needed in the D-Cache.

EXtreme Virtual Pipelining (XVP) takes a step toward scalable, multithreaded processors by avoiding the pitfalls constraining conventional designs. Instead of increasing the size of critical path resources and attempting to learn optimal allocations, XVP chooses to virtualize pipeline resources, to provide mechanisms for those resources to dynamically partition themselves, and to add a 3rd L1-Cache for storage of those resources. Future versions of XVP could virtualize other non-critical path shared resources like branch predictors, branch target buffers or load-wait tables. Additionally, XVP's virtualization methods can be used to optimize single-threaded processors by providing the illusion of more pipeline resources than is traditionally available on a single-threaded processor.

3. REFERENCES

[1] Oehmke, D. W., Binkert, N. L., Mudge, T., and Reinhardt, S. K. 2005. How to Fake 1000 Registers. *Proceedings of the 38th Annual IEEE/ACM international Symposium on Microarchitecture* , 2005.