Summary Part 2
Continual Flow Pipelines

Mrinal Nath (ID: 3307043)
April 20, 2005

Did this paper address an important issue? Explain.

To achieve high ILP, conventional (i.e. ROB-based) processors need to have large physical register files and schedulers. This is mainly because, conventionally, these structures are ‘blocking’ – operands of a long latency instruction (e.g. a load that missed in L2) continue to occupy entries in these structures during the latency period. The limited number of these resources produces stalls due to structural hazards. However, increasing these structures beyond a certain limit produces diminishing returns, since these large structures directly increase the cycle time, power consumption and area of the processor core. This also makes such cores less suitable for integrating into CMPs.

This paper proposes CFPs which can avoid the structural stalls by providing non-blocking register files and scheduler. This allows the processor to maintain a very large instruction window and extract large amounts of ILP, without requiring large register files, large schedulers or large L2 caches. This allows for a smaller processor core with a faster clock and lower power consumption, while achieving higher performance than the conventional processors. CFP-based processors cores are also more suitable for building CMPs.

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

CFP builds on the CPR proposal by addressing the resource inefficiency (blocking scheduler) in CPR. The basic idea is to force the long latency instructions (loads that miss in L2) and their slices to drain out of the pipeline into a special hardware unit called the Slice Processing Unit, where these instructions wait till all their input operands are available. Thus, the resources occupied by these instructions can be freed up and made available to the instructions that are not in the slice (independent of the long latency instruction). The slice instructions are reintroduced into the pipeline when their operands are available. This is a valid technique.

Strengths

- Supports very large instruction windows and achieves high ILP while not requiring large register files and scheduler. This also reduces the area and power requirements
- The scheduler design need not change – existing designs can be reused
- Requires minimal changes to the existing CPR core (e.g. NAV bit)
- The slice instructions carry the ready operand values with them into the SPU, allowing the completed source registers to be freed up
- Back-end (physical-to-physical) renaming is used so that dependent destination registers can be freed up
- Performs better than conventional processors even with much smaller L2 caches
- Attractive building block for CMPs
- Allows sizing of register files and scheduler to be independent of the ‘desired’ instruction window
• Bridges the gap between ideal CPR and practical CPR performance by eliminating the blocking scheduler used in practical CPR
• Lot of useful work is done in the shadow of the long latency instructions – unlike other approaches (e.g. run-ahead execution), this work is not discarded

**Weaknesses**

• Exposes branch prediction accuracy as the primary performance bottleneck – requires highly accurate branch prediction for achieving high performance
• There are special cases (live-outs) where the back-end renaming cannot be used. Overall, the SPU introduces new complexity and requires thorough validation

**Do the results support the conclusions? Explain.**

The main conclusions are that CFP based processors can sustain large instruction windows and extract large amount of ILP while keeping register file, scheduler and L2 cache small. Also, CFP is an attractive building block for CMPs due to lower area and power requirements.

The results amply support these conclusions. CFP can close the gap between ideal CPR and practical CPR on all types of workloads by solving the resource inefficiency problem (blocking scheduler) of CPR. CPR always performs better than ROB – CFP performs better than both of them (4% to 46% speed up over ROB for various workloads). It can sustain large instruction windows (more than 2048 instructions for 20% to 40% of the execution time for various workloads). It increases the amount of MLP by sustaining more outstanding misses. It reduces the need for large L2 caches, since it performs better than ROB even with smaller L2 caches. It potentially reduces power consumption, since the size of active structures need not be large. Also, smaller L2 cache directly reduces power wastage due to leakage. Due to these reasons, CFP can achieve high performance for single threads, while being useful for building CPMs to help increase throughput.

**Describe the potential future works?**

• Analysis of the power consumption – whether it reduces due to above mentioned reasons, or increases due to increased wrong-path execution
• Branch predictors that are more accurate than existing ones – so that the gap between ideal CPR and CFP can be closed even further (some of the area saved can be dedicated to bigger and hopefully better branch predictors). This may also help the checkpoint mechanism be more accurate and reduce the number of checkpoints needed
• More aggressive register reclamation mechanisms