Critique of paper “Multiplex: Unifying Conventional and Speculative Thread-Level Parallelism on a Chip Multiprocessor”[1]

Name: Renwei Yu
ID: 1200859378

In this paper, the authors mention that there are two forms of TLP for Chip multiprocessors (CMPs): explicit threading and implicit threading. However, explicit threading is hard to program manually and, if automated, is limited in performance due to serialization of unanalyzable program segments. Implicit threading, on the other hand, requires buffering of program state to handle misspeculations, and is limited in performance due to buffer overflow in large threads and dependences in small threads. So this paper propose the Multiplex architecture to unify implicit and explicit threading in CMPs to combine the complementary strength of implicit and explicit threading to alleviate the individual weakness of these two schemes. The following are my comments on this paper:

Strengths:

- The authors present architecture (hardware and compiler) mechanisms for selection, dispatch, and data communication to unify explicit and implicit threads from a single application.

- The authors propose the Multiplex Unified Coherence and Speculative versioning (MUCS) protocol that provides unified support for coherence in explicit and implicit threads of a single application executing on multiple cores with private cores.

- The paper shows that neither an implicitly-threaded nor explicitly-threaded architecture performances consistently better across the benchmarks, and for several benchmarks there is a large performance gap between the two architectures. Multiplex matches or outperforms the better of the two architectures on every benchmark by 16% on average.

Weakness:

- The authors do not consider about the overhead when switch from implicit mode to explicit mode or vice versa. For example, the Multiplex will suspend dispatch upon mode switch to allow thread commits to complete.

- In the Section 3.1, Thread Selection, the authors mention that the Multiplex extracts parallelism from implicit threads at runtime with the help of hardware speculation, but they do not give the details of how the hardware can achieve this.

- In the thread dispatch part of Multiplex, the authors mention that it will assign a thread from the program to execute on a CPU. But what if the thread migrate from the this original CPU to others after some time? How to maintain the state for that thread? The authors seem do not consider about this issue.
On the whole the paper gives a creative and novel architecture that can combine the strengths of both implicit and explicit threading to avoid the individual weakness of the two schemes.