Chip multiprocessors (CMPs) exploit thread-level parallelism (TLP), and are displacing traditional superscalar architectures. The authors proposed the Multiplex architecture for CMPs to unify implicit and explicit threading. They presented architectural (hardware and compiler) mechanisms for selection, dispatch, and data communication from a single application. They also proposed the Multiplex Unified Coherence and Speculative versioning (MUCS) protocol that provided unified support for coherence in explicit threads and speculative versioning in implicit threads of a single application executing on multiple cores with private caches.

**Strengths:**

1. Multiplex on average achieves a speedup of 2.69, improving speedups by 16% over explicit-only and 22% over implicit-only CMPs. In 7 out of 11 applications, Multiplex improves speedups over the better of the two on average by 10%
2. Multiplex exploits the similarities to allow efficient implementation without much extra hardware. It also makes use of “best of both worlds” approach. It uses complementary strengths of implicit and explicit threading to alleviate the individual weaknesses of the two schemes.
3. The key to maintaining correctness in both modes during data communication is the MUCS protocol which tracks the copies and versions of every cache block present in the system. Explicit mode - MUCS tracks the location of copies in the system and takes appropriate action on loads and stores. Implicit mode - MUCS tracks both the location and the program order among the versions.
4. Multiplex increases opportunity for eliminating thread dependence by executing compile time analyzable program segments as explicit threads.
5. Explicit-only CMPs use fine-grain threads when the compiler cannot analyze dependences among larger thread bodies. Multiplex can execute these fine-grain threads as implicit threads, thereby reducing the thread dispatch/completion overhead.
6. The data speculation overhead by execution in independent threads as explicit threads is reduced by Multiplex. So there is no need for speculation because of Multiplex.

**Weaknesses:**

1. The paper suggests that compiler has the opportunity to choose between implicit and explicit threading models to maximize performance on a per program and per program segment basis. However, there are no metrics provided to substantiate if the latency to choose between these 2 threading models would not be detrimental as the number of processors increase.
2. Thread size is a key factor affecting performance in both explicit-only and implicit-only CMPs. Smaller thread sizes will decrease the scope of parallelism, increase likelihood of data dependence across threads and the impact of thread dispatch/completion overhead.
3. Compiler is left to do a lot of work in this proposed architecture scheme. They haven’t discussed any algorithms that can be/have been implemented in compilers that would support this scheme (Polaris, SUIF). For example – they suggest that their heuristics-based compiler optimizations make a near-optimal decision in choosing between explicit and implicit threads. They haven’t elaborated on these optimizations.
4. The paper does not discuss schemes for software changes that must be done to inform the hardware which type of threading is used for a given program segment so that hardware can provide the appropriate execution support.
5. Control flow irregularities impacts performance in Multiplex for program segments that execute as implicit threads.