

# Relativistic Programming

Josh Triplett  
Portland State University  
Student; Presenter

Paul E. McKenney  
IBM Linux Technology Center

Phil Howard  
Portland State University  
Student

Jonathan Walpole  
Portland State University

We introduce “Relativistic Programming” (RP), a concurrent programming technique with much practical success but which has only recently become the subject of formal study. RP algorithms are distinguished by two properties. The first property is that RP algorithms tolerate different threads seeing events occurring in different orders, so that events are not necessarily globally ordered, but rather subject to constraints of per-thread ordering, and in a few cases, partial-order constraints on global ordering. The second property is that RP algorithms tolerate conflicts; for example, one thread can safely modify a memory location despite the fact that other threads might be concurrently reading that same memory location.

These properties allow several important classes of problems to be addressed with wait-free linearly scalable algorithms having overheads rivaling that of the equivalent synchronization-free single-threaded code. In many cases, these algorithms have extremely simple implementations, so that practitioners can readily make use of them. A case in point is read-copy update (RCU), which was accepted into the Linux kernel in October 2002, and as of June 2009 has more than 2500 uses in the 2.6.30 Linux kernel.

The benefits of RP arise primarily from this tolerance for reordering and conflicts, because global ordering and conflict-avoidance defeat many critical compiler and hardware optimizations. Without these optimizations, frequent global agreement is required, which introduces unavoidable delays imposed by the communication latency between processors. No amount of hardware optimization can eliminate these delays without some software tolerance for reordering and conflicts.

To provide these tolerance properties, a Relativistic Programming framework supplies the means for an RP algorithm to express specific constraints on conflicts and reordering as needed for correctness, leaving the compiler and the hardware free to aggressively optimize other portions of the algorithm as they see fit. The practice of RP consists of using combinations of these constraints to obtain simple, fast, and scalable concurrent programs.