

4.1. SIMT Architecture

The multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads called *warps*. Individual threads composing a warp start together at the same program address, but they have their own instruction address counter and register state and are therefore free to branch and execute independently. The term *warp* originates from weaving, the first parallel thread technology. A *half-warp* is either the first or second half of a warp. A *quarter-warp* is either the first, second, third, or fourth quarter of a warp.

When a multiprocessor is given one or more thread blocks to execute, it partitions them into warps and each warp gets scheduled by a *warp scheduler* for execution. The way a block is partitioned into warps is always the same; each warp contains threads of consecutive, increasing thread IDs with the first warp containing thread 0. [Thread Hierarchy](#) describes how thread IDs relate to thread indices in the block.

A warp executes one common instruction at a time, so full efficiency is realized when all 32 threads of a warp agree on their execution path. If threads of a warp diverge via a data-dependent conditional branch, the warp executes each branch path taken, disabling threads that are not on that path. Branch divergence occurs only within a warp; different warps execute independently regardless of whether they are executing common or disjoint code paths.

The SIMT architecture is akin to SIMD (Single Instruction, Multiple Data) vector organizations in that a single instruction controls multiple processing elements. A key difference is that SIMD vector organizations expose the SIMD width to the software, whereas SIMT instructions specify the execution and branching behavior of a single thread. In contrast with SIMD vector machines, SIMT enables programmers to write thread-level parallel code for independent, scalar threads, as well as data-parallel code for coordinated threads. For the purposes of correctness, the programmer can essentially ignore the SIMT behavior; however, substantial performance improvements can be realized by taking care that the code seldom requires threads in a warp to diverge. In practice, this is analogous to the role of cache lines in traditional code: Cache line size can be safely ignored when designing for correctness but must be considered in the code structure when designing for peak performance. Vector architectures, on the other hand, require the software to coalesce loads into vectors and manage divergence manually.