Seit C++11 stellt sich C++ den Anforderungen der Multicore-Architekturen. Der 2011 veröffentlichte Standard definiert, wie sich ein C++ Programm bei mehreren Threads zu verhalten hat. Dabei setzen sich die C++11 Multithreading-Fähigkeiten aus zwei Komponenten zusammen. Das ist zum einen das definierte Speichermodell, das ist zum anderen die standardisierte Threading-Schnittstelle.

Mit C++17 wird es parallele Versionen fast aller Algorithmen der Standard Template Library geben. So kann einem Algorithmus die sogenannten execution policy mitgegeben werden. Die execution policy bestimmt, ob der Algorithmus sequentiell (std::seq), parallel (std::par) oder parallel und vektorisierend (std::par_unseq) ausgeführt wird. So wird die erste und zweite Variante des sort Algorithmus sequentiell, die dritte parallel und die vierte parallel vektorisierend ausgeführt. C++20 bietet ganz neue Multithreading Konzepte in C++ an. Diese zeichnen sich im wesentlichen dadurch aus, dass sie Multithreading einfacher und damit weniger fehleranfällig machen.

Die atomaren Smart Pointer std::shared_ptr und std::weak_ptr besitzen ein konzeptionelles Problem in Multithreading Programmen. Sie teilen ihren veränderlichen Zustand. Damit sind sie natürlich implizit der Gefahr von kritischen Wettläufen und damit von undefiniertem Programmverhalten ausgesetzt. Zwar sichern std::shared_ptr und std::weak_ptr zu, dass das Inkrementieren und Dekrementieren der Referenzzähler eine atomare Operation ist und dass der Destruktor der Ressource genau nur einmal aufgerufen wird, aber sie sichern nicht zu, dass die Zugriffe auf ihre Ressourcen atomar sind. Damit räumen die neuen atomaren Smart Pointer auf. Mit Tasks in der Form von Promisen und Futuren führte C++11 ein neues Multithreading Konzept in C++ ein. Trotz ihres großen Mehrwertes besitzen sie eine große Unzulänglichkeit. Futures in C++11 können nicht komponiert werden.