Of course, OpenMP is not the only way of tweaking C...
Cilk Plus

Of course, OpenMP is not the only way of tweaking C

*Cilk Plus* is somewhat similar in that it adds annotations and is based on fork and join
Of course, OpenMP is not the only way of tweaking C

*Cilk Plus* is somewhat similar in that it adds annotations and is based on fork and join

But as new keywords in C, not as pragmas (mostly)
Of course, OpenMP is not the only way of tweaking C

_Cilk Plus_ is somewhat similar in that it adds annotations and is based on fork and join

But as new keywords in C, not as pragmas (mostly)

_Cilk Plus_ is intended as an extension to C++, but works for C, too
Cilk Plus

Of course, OpenMP is not the only way of tweaking C

*Cilk Plus* is somewhat similar in that it adds annotations and is based on fork and join

But as new keywords in C, not as pragmas (mostly)

Cilk Plus is intended as an extension to C++, but works for C, too

You may come across other versions named “Cilk” and “Cilk++”
Cilk Plus

Cilk Plus has a task-based view of computation (like TBB), rather than thread based.
Cilk Plus has a task-based view of computation (like TBB), rather than thread based.

This means the programmer thinks about what tasks need to be done, and Cilk Plus thinks about the best way of assigning those tasks to threads.
Cilk Plus

Cilk Plus has a task-based view of computation (like TBB), rather than thread based.

This means the programmer thinks about what tasks need to be done, and Cilk Plus thinks about the best way of assigning those tasks to threads.

And similar to OpenMP, the number of threads used and the threading mechanisms are mostly hidden from the programmer.
int fib (int n) {
    if (n < 2) return n;
    else {
        int x, y;

        x = cilk_spawn fib(n-1); // fork
        y = fib(n-2);
        cilk_sync; // join
        return (x+y);
    }
}

(from the Cilk Plus website)
Cilk Plus

- Cilk Plus has just three main keywords: cilk_spawn, cilk_sync and cilk_for
Cilk Plus

- Cilk Plus has just three main keywords: cilk_spawn, cilk_sync and cilk_for
- So is much simpler than OpenMP
Cilk Plus

- Cilk Plus has just three main keywords: cilk_spawn, cilk_sync and cilk_for
- So is much simpler than OpenMP
- And more lightweight to use
Cilk Plus

- Cilk Plus has just three main keywords: cilk_spawn, cilk_sync and cilk_for
- So is much simpler than OpenMP
- And more lightweight to use
- And seemingly less flexible: but Cilk Plus provides other mechanisms for more advanced control
Cilk Plus

- Cilk Plus has just three main keywords: cilk_spawn, cilk_sync and cilk_for
- So is much simpler than OpenMP
- And more lightweight to use
- And seemingly less flexible: but Cilk Plus provides other mechanisms for more advanced control
- Ignoring the keywords leaves a valid equivalent sequential C program
Cilk Plus

- Cilk Plus has just three main keywords: cilk_spawn, cilk_sync and cilk_for
- So is much simpler than OpenMP
- And more lightweight to use
- And seemingly less flexible: but Cilk Plus provides other mechanisms for more advanced control
- Ignoring the keywords leaves a valid equivalent sequential C program

A cilk_for indicates a parallelisable for loop
Cilk Plus

- Cilk Plus has just three main keywords: `cilk_spawn`, `cilk_sync` and `cilk_for`
- So is much simpler than OpenMP
- And more lightweight to use
- And seemingly less flexible: but Cilk Plus provides other mechanisms for more advanced control
- Ignoring the keywords leaves a valid equivalent sequential C program

A `cilk_for` indicates a parallelisable `for` loop

There is an implicit `cilk_sync` at the exit of every function that contains a spawn
Cilk Plus also employs work stealing of tasks, but in a more subtle way than TBB.
Cilk Plus also employs work stealing of tasks, but in a more subtle way than TBB

In the code

cilk_spawn fun1();
fun2();
the *current* thread actually starts executing fun1()
Cilk Plus

In more detail:

• when the current thread reaches the cilk spawn it saves the current continuation (i.e., the point in the code just before the fun2()) on its continuation stack
• it then starts executing fun1()
• when done with that, it pops the continuation stack and starts executing what it finds there
Cilk Plus

In more detail:

- when the current thread reaches the `cilk_spawn` it saves the current continuation (i.e., the point in the code just before the `fun2()`) on its continuation stack.
In more detail:

• when the current thread reaches the cilk_spawn it saves the current continuation (i.e., the point in the code just before the fun2()) on its continuation stack

• it then starts executing fun1()
In more detail:

- when the current thread reaches the `cilk_spawn` it saves the current continuation (i.e., the point in the code just before the `fun2()`) on its continuation stack
- it then starts executing `fun1()`
- when done with that, it pops the continuation stack and starts executing what it finds there
An idle other thread can steal a continuation and start executing it
Cilk Plus

An idle other thread can steal a continuation and start executing it

Thus leading to the initially surprising behaviour that `fun2()` might get stolen, not `fun1()`
Cilk Plus

An idle other thread can steal a continuation and start executing it

Thus leading to the initially surprising behaviour that `fun2()` might get stolen, not `fun1()`

In contrast with TBB, where the current thread pushes `fun1()` and so it is that that can be stolen
Cilk Plus

An idle other thread can steal a continuation and start executing it

Thus leading to the initially surprising behaviour that \texttt{fun2()} might get stolen, not \texttt{fun1()}

In contrast with TBB, where the current thread pushes \texttt{fun1()} and so it is that that can be stolen

TBB implements \textit{child stealing}; Cilk Plus has \textit{continuation stealing}
Manipulating continuations is why Cilk Plus needs compiler support. Child stealing as implemented by TBB is implementable in C++ directly as it is essentially just pushing and popping functions on a queue.
Manipulating continuations is why Cilk Plus needs compiler support. Child stealing as implemented by TBB is implementable in C++ directly as it is essentially just pushing and popping functions on a queue.

The difference is that continuation stealing has better memory use patterns than the child stealing and so tends to give better parallelism.
Manipulating continuations is why Cilk Plus needs compiler support. Child stealing as implemented by TBB is implementable in C++ directly as it is essentially just pushing and popping functions on a queue.

The difference is that continuation stealing has better memory use patterns than the child stealing and so tends to give better parallelism.

Exercise. Child stealing can have unlimited memory use, while continuation stealing does not. Read about this.
Cilk Plus

Whatever the relative merits, OpenMP and Thread Building Blocks have wide recognition while Cilk Plus is quite niche
Cilk Plus

Whatever the relative merits, OpenMP and Thread Building Blocks have wide recognition while Cilk Plus is quite niche.

In fact, Intel now has deprecated Cilk Plus in favour of their TBB, which being a purely library-based mechanism is easier to support, despite being slightly worse in behaviour.
Cilk Plus

Whatever the relative merits, OpenMP and Thread Building Blocks have wide recognition while Cilk Plus is quite niche.

In fact, Intel now has deprecated Cilk Plus in favour of their TBB, which being a purely library-based mechanism is easier to support, despite being slightly worse in behaviour.

Exercise. Read about the many other parts of Cilk Plus, such as vector sections.
Cilk Plus

Whatever the relative merits, OpenMP and Thread Building Blocks have wide recognition while Cilk Plus is quite niche.

In fact, Intel now has deprecated Cilk Plus in favour of their TBB, which being a purely library-based mechanism is easier to support, despite being slightly worse in behaviour.

Exercise. Read about the many other parts of Cilk Plus, such as vector sections.

Exercise. Work through how continuation stealing might execute the `parallel_for` example.
Whatever the relative merits, OpenMP and Thread Building Blocks have wide recognition while Cilk Plus is quite niche.

In fact, Intel now has deprecated Cilk Plus in favour of their TBB, which being a purely library-based mechanism is easier to support, despite being slightly worse in behaviour.

Exercise. Read about the many other parts of Cilk Plus, such as `vector sections`.

Exercise. Work through how continuation stealing might execute the `parallel_for` example.

Exercise. Compare Cilk Plus and TBB.