Why is std::hardware_destructive_interference_size a compile-time constant instead of a run-time value?

devblogs.microsoft.com/oldnewthing/20230424-00

April 24, 2023

Raymond Chen

C++17 added a new compile time constant `std::hardware_destructive_interference_size` which tells you (basically) the size of a cache line. The purpose of this is to allow you to lay out your structures in a way that avoids false sharing.[1]

But how does the compiler know what the cache line size will be of the CPU the program will eventually be run on? Shouldn't this be a run-time value instead of a compile-time value?

Well yes, the actual size of the cache line isn't know until run-time, because it is only then that the program meets a CPU. But really, you want this to be a compile-time constant, even if it's the wrong compile-time constant.

Structure layouts are determined at compile time, and you're going to be using `std::hardware_destructive_interference_size` as part of a `alignas()`, which requires a compile-time constant.

So the compiler picks a "most likely" value for you to optimize for.

If you really want your code to adapt to the run-time cache size, you can generate multiple versions of the structure and choose among them at run-time.

```
template<std::size_t alignment>
struct two_things
{
    alignas(alignment) std::atomic<int> evens;
    alignas(alignment) std::atomic<int> odds;
};

template<typename alignment>
void do_important_multithreaded_calculations_at_alignment()
{
    ... two_things<alignment> ...
}

void do_important_multithreaded_calculations()
{
    // operating system-specific code to get cache line size
    auto alignment = get_runtime_cache_line_size();

    if (alignment <= 32) {
        do_important_multithreaded_calculations_at_alignment<32>();
    } else if (alignment <= 64) {
        do_important_multithreaded_calculations_at_alignment<64>();
    } else {
        do_important_multithreaded_calculations_at_alignment<128>();
    }
}
```

So how does the compiler choose the value to report at compile time?

In a way, you told the compiler what value to use.

Based on the compiler command line options, you might be telling it to “optimize for Intel Pentium” or “optimize for Intel Skylake” or “optimize for AMD K8.” The constant should reflect the cache line size for the processor family you specified. If you end up not running on that CPU, then your choices may end up suboptimal, but at least you’ll still get the right answer, just perhaps not in the fastest way.

This may sound like a bummer, until you realize that the compiler is already making this trade-off behind your back!

When choosing what code to generate, the compiler is already making assumptions about the CPU it will be targeting. Depending on the CPU, the optimal code sequence may vary. For example, on Pentium, you want to order the instructions so that the two internal CPU pipes are both busy as much as possible. The optimal amount of loop unrolling may vary depending on the CPU. The optimal placement of jump targets (to take advantage of branch target buffers, branch prediction, and instruction fetching) may vary from CPU to CPU. Even the best type of return instruction can vary. And what is optimal for one CPU may be suboptimal for another.

The compiler is making these types of decisions all the time. It's just that in the case of the cache line, it's letting you see it.

[1] There is a counterpart constant `std::hardware_constructive_interference_size` which also specifies the size of a cache line, but this value is for laying out your structures so that you can force values to share a cache line because they are used together. You can read the full proposal for more details.