C++ Club UK Meeting 143

Gleb Dolgich

2022-02-03

Contents

Why do you like C++?

An amusing thread on Reddit.

Some replies:

> I like it because it does what I ask it to do. And also because it does not do things I didn't ask it to do #

> Philosophy: "Programmers should be free to pick their own programming style, and that style should be fully supported by C++." #

> I like it because it makes you say "wtf why?" at compile time rather than runtime. #

> I don't like C++ except for a singular reason - my employer pays me well for writing C++. #

2021-01 ISO C++ mailing

Mailing, Reddit

`const` all the things?

Arthur O'Dwyer wrote this article on his blog. In it he lists places where he uses `const` and where he doesn't:

Const:

> In function signatures: passing by `const` reference, `const` member functions

No const:

> In function signatures: passing by value Data members: never `const` > <...> the point of making a class with private members is to preserve invariants among those members. "This never changes" is just one possible invariant. Some people hear "never changes" and think it sounds a bit like `const`, so they slap const on that data member; but you shouldn't lose sight of the fact that the way we preserve invariants in C++ isn't with `const`, it's with `private`. Return types: never `const`

Rarely `const`:

> Local variables (*I tend to disagree*)

Reddit thread. A few redditors think that locals should be as `const` as possible.

This is another Reddit thread on making local variables `const`, and most commenters there agree with that.

Best CPU vs GPU: 879 GB/s Reductions in C++

This article iterates through various methods to speed up calculations in C++ using vectorization and parallelization on CPU and GPU. The author provides code snippets that add 1GB floating numbers together and their throughput data. Let's see what are the results.

- Plain C++ and STL: around 5.2–5.3 GB/s
- SIMD AVX2 using intrinsics: 17–22 GB/s
- OpenMP: 5.4 GB/s
- Parallel STL based on Intel Threading Building Blocks (TBB): 80–87 GB/s
- SIMD + Threads: 89 GB/s
- CUDA: 817 GB/s
- Thrust: 743 GB/s (nice code!)

```
thrust::reduce(numers.begin(), numbers.end(), float(0),
    thrust::plus<float>());
```

- CUB: 879 GB/s

This is from Thrust home page:

> Thrust is a parallel algorithms library which resembles the C++ Standard Template Library (STL). Thrust's high-level interface greatly enhances programmer productivity while enabling performance portability between GPUs and multicore CPUs. Interoperability with established technologies (such as CUDA, TBB, and OpenMP) facilitates integration with existing software.

Of all the above, based on the presented code snippets, I would choose Thrust.