CPP Con 2019 Trip Report (Overview)

CppCon has just ended(Edit: it's been ~ month since :)). As I sit cramped in this tiny airplane seat, I try to process it all (yes! pretend like you wrote this on the plane ;)). While a lot of the things made sense during the talks, my notes look like a jumble of loosely connected keywords. Let's try to do reverse debugging here ...

Venue: Gaylord Rockies Resort, Denver, Co. It was a massive and pretty impressive hotel in middle of no where.

Broad Topics

1. Back to Basics (new track)
2. Concurrency
3. GPU/Graphics
4. Compilers/Tooling
5. Data Structures/Allocation
6. Design/Best Practices
7. Future of C++
8. Security/Safety

Most Interesting Talks

The Best Parts of C++ - Jason Turner

This was introduction to Back to Basics track.

This is a fun one which discusses evolution and advantages of major C++ features.

Best parts of C++:

0. C++ standard (since C++98)
1. `const` (helps simplify variable declarations and write clean code. Forces user to initialize values)
2. RAII (since C++98, e.g. ctors/dtors pair with scoped values)
3. Templates (Do not repeat yourself, Turing complete and do not need *type-erasure*)
4. `vector` containers (takes care of *copy elison*)
5. Algo & STL (e.g. `for_each`, `any_of`)
6. `std::array` (fixed size at compile time -> more crazy optimization)
7. List initialization (c++11)
8. Variadic Templates (critical for implementing std::function)
9. `constexpr` (since c++11, reduced restrictions in c++20)
10. `auto` (since C++11,)
11. `auto` Return type deduction (since c++14, allows for creation of generic `lambda`s)
12. Lambda (since c++11)
13. Generic and variadic lambda (since c++14)

14. range based for loop (since c++11, `for (const auto))
15. structured bindings (c++17, `const auto &[e1, e2] = something`)
16. Concepts (C++20, simplifies writing templated / `SFINAE` code , can defined easily from type traits)
17. `string_view` (C++17, cheaper than copying strings from const char*)
18. Text Formatting (some nice features e.g. positional,named args (python like) coming C++20,)
19. ranges (allows piping operations, C++ 20)
20. CTAG (class template argument deduction, gian impact for arrays C++17)
21. Rvalue references (C++11)
22. Guaranteed Copy Elision (C++17, compilers have done it historically)
23. Defaulted and Deleted functions (C++ 11)
24. `std::unique_ptr`(c++11) (safety by default)
25. fold expressions (for variadic expansions, C++17)

[Speed is found in the minds of people - Andrei Alexandrescu]()

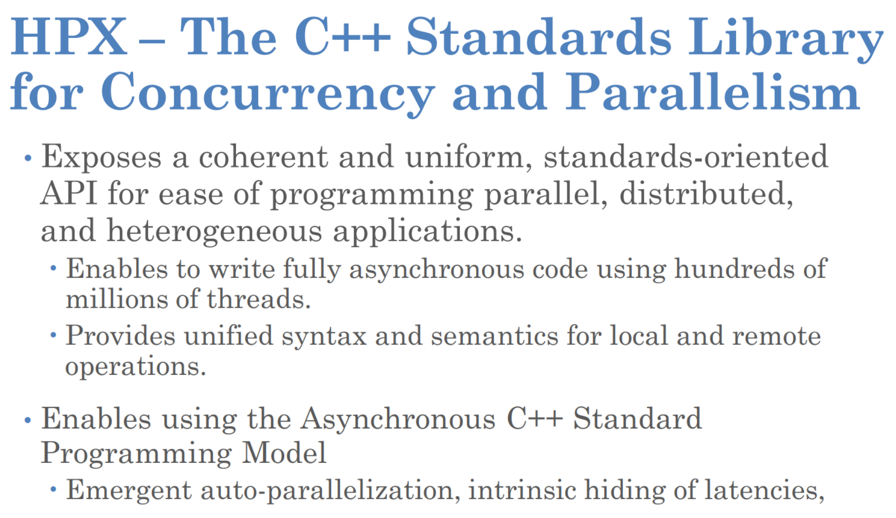
This was extremely entertaining and insightful. It shows why creativity is important part when optimizing software. Andrei Alexandrescu talks about his experiments with optimizing sorting for small datasets.

Key Takeaways:

1. First level thinking:
    1. Try Silly things - Andrei's sorting algorithm does more work (create a heap before doing insertion sort) and special tricks for last element , but still is better than standard algorithms for small data sets.
    2. Question the metrics used for designing the algorithm. [Parametric complexity theory]()
2. Second level thinking: Generic Programming can prevent from achieving performance. Related paper [P1393: A General Property Customization Mechanism]()

[Asynchronous Programming in Modern C++ - Hartmut Kaiser]()

Hartmut Kaiser talks about [HPX]()



HPX – The C++ Standards Library for Concurrency and Parallelism

- Exposes a coherent and uniform, standards-oriented API for ease of programming parallel, distributed, and heterogeneous applications.
    - Enables to write fully asynchronous code using hundreds of millions of threads.
    - Provides unified syntax and semantics for local and remote operations.
- Enables using the Asynchronous C++ Standard Programming Model
    - Emergent auto-parallelization, intrinsic hiding of latencies,

SG14 Meeting

Researches from Sandia National Labs presented their paper [A free function Linerar algebra interface based on the BLAS] for C++23.

Interesting highlights

- This paper takes advantage of `mdspan` paper which defines types and functions for mapping multidimensional indices and `mdarray`
- There is some talk about optionally providing "batched" gemm interfaces. Batched gemm standardization effort http://icl.utk.edu/bblas/
- Mixed precision computation supported.
- "Like the C++ Standard Library's algorithms, our operations take an
  optional execution policy argument. This is a hook to support
  parallel execution and hierarchical parallelism (through the
  proposed executor extensions to execution policies, see
  Integrating Executors with Parallel Algorithms - P1019R2)"

C++ ABI from ground up - Louise Dionne

Louise Dionne who is developer of Apple's std library gives an interesting overview of how ABI compatibility can break. He gives example of Unix + Itanium C++ ABI , but the issues are universal. ABI stability is important for

- distributing binary without source
- plug and play binaries in your code without rebuilding.

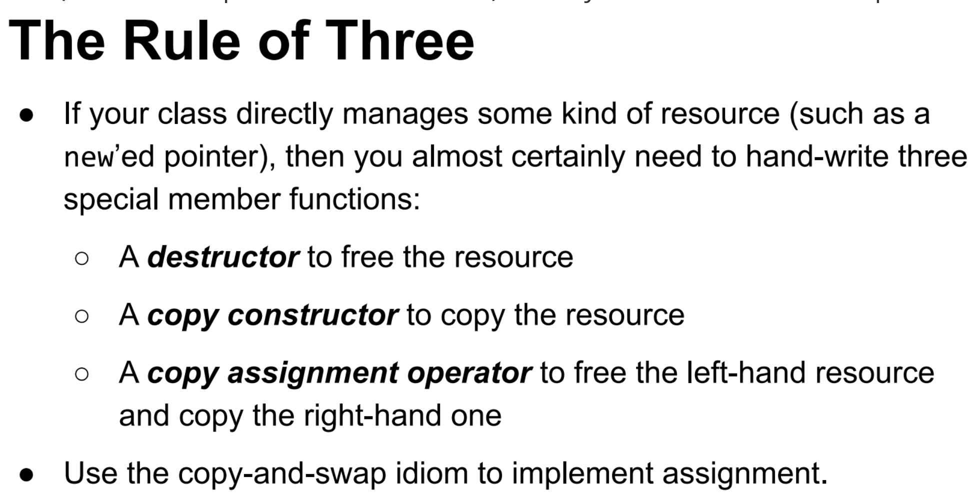
RAII and Rule of Zero - Arthur O'Dwyer

This is a really useful introduction to resource management. Arthur O' Dwyer is amazing at explaining

advanced concepts.

Key Takeaways:

1. RAII (Resource Acquistion is Initialization) is really about resource cleanup.



The Rule of Three

- If your class directly manages some kind of resource (such as a new'ed pointer), then you almost certainly need to hand-write three special member functions:
  - A ***destructor*** to free the resource
  - A ***copy constructor*** to copy the resource
  - A ***copy assignment operator*** to free the left-hand resource and copy the right-hand one
- Use the copy-and-swap idiom to implement assignment.

2.

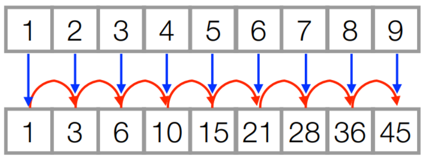
Related Talks

RVO - Arthur O'Dwyer

Initialization in Modern C++

From Algorithm to Generic Parallel Code : Diemeter Kuhl

Dimeter Kuhl talks about his purely curiosty driven study on parallelizing inclusive scan using OpenMP and TBB. *Inclusive scan*: this seemingly simple (sequentially O(n)) operation is tricky to parallelize since each result depends on the previous.



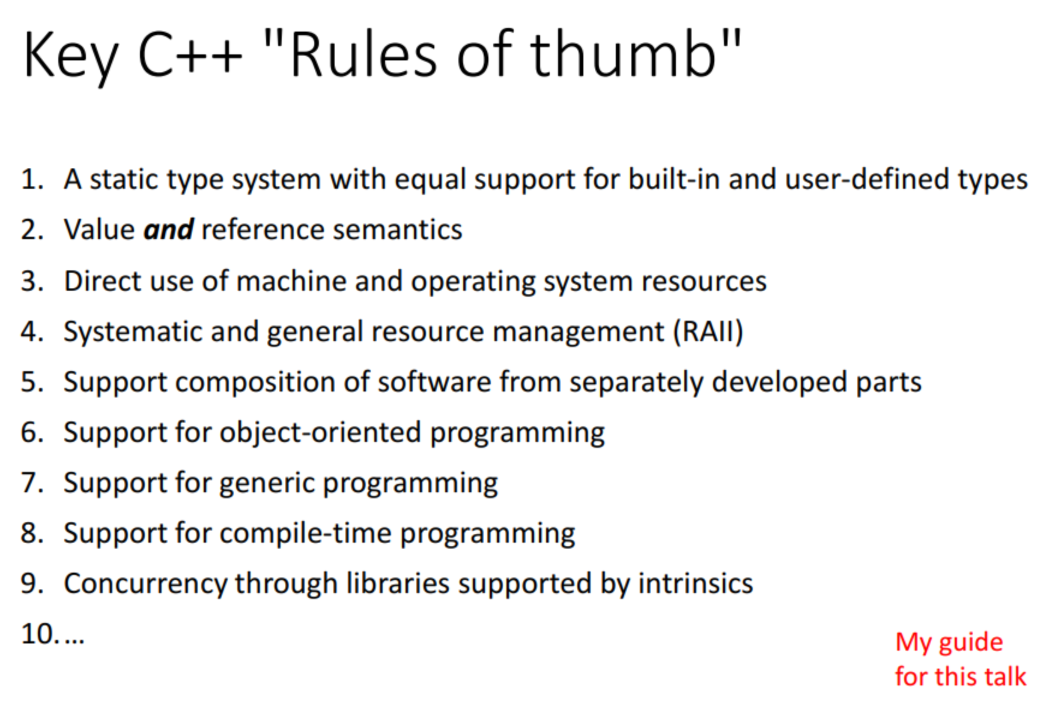
Parallel algorithm of inclusive scan divides the data into chunks and partial sums of these chunks are computed in parallel. Performance is compared between standard algorithms, OpenMP and TBB on an Intel Xeon Phi.

Other Talks

C++20: C++ at 40 : Bjarne Stroustrup

"If you do it right, nobody can see it", Bjarane Stroustrup.

Bjarne Stroustrup in his supremely humble way talks about design principles of C++.



Key C++ "Rules of thumb"

1. A static type system with equal support for built-in and user-defined types
2. Value ***and*** reference semantics
3. Direct use of machine and operating system resources
4. Systematic and general resource management (RAII)
5. Support composition of software from separately developed parts
6. Support for object-oriented programming
7. Support for generic programming
8. Support for compile-time programming
9. Concurrency through libraries supported by intrinsics
10. …

My guide
for this talk

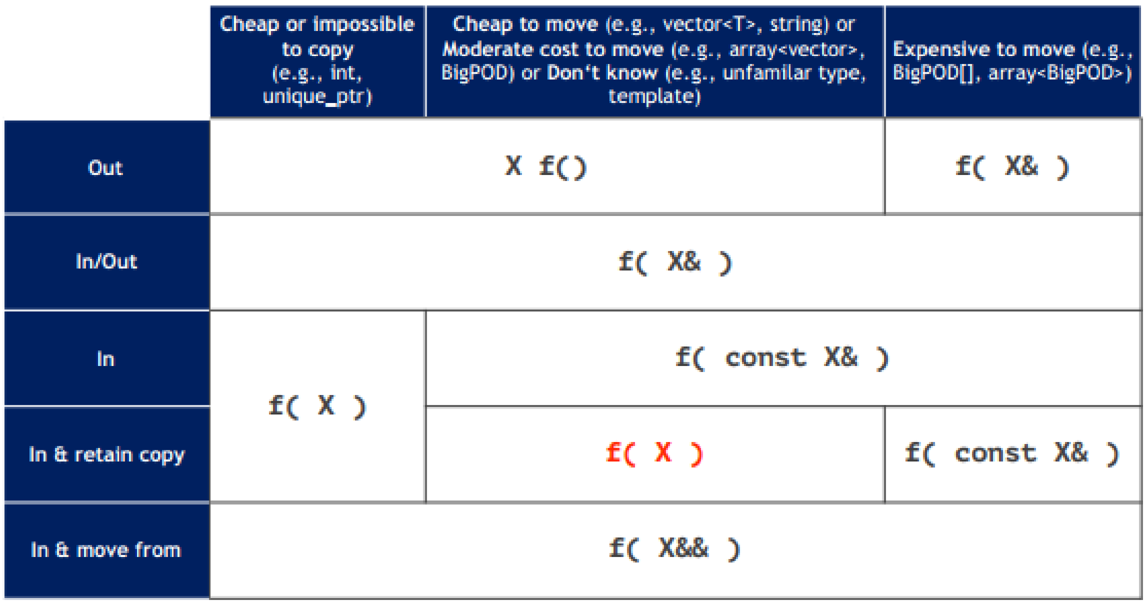
Move Sematics: Klaus Iglberger (Part 1 & 2)

Klaus Iglberger gives an in-depth and engaging introduction to l-values, r-values, `std::move` and `srd::forward`. The most fun part were the quizzes for the audience in between each concept which clarified subtle points.

"... I think the most important take-away is that programmers should be leery of following patterns without thought. ..." (Howard Hinnant)

Key Takeaways:

1. Prefer simple ways of passing information



	Cheap or impossible to copy (e.g., int, unique_ptr)	Cheap to move (e.g., vector<T>, string) or Moderate cost to move (e.g., array<vector>, BigPOD) or Don't know (e.g., unfamilar type, template)	Expensive to move (e.g., BigPOD[], array<BigPOD>)
Out	`X f()`		`f( X& )`
In/Out	`f( X& )`		
In	`f( X )`	`f( const X& )`	
In & retain copy		`f( X )`	`f( const X& )`
In & move from	`f( X&& )`		

2. **Rule of zero**: (Core guideline C2.0) If you can avoid defining default operations, do.
3. [TODO]Return Value optimization and antipatterns

Behind Enemy Lines - Reverse Engineering C++ in Modern Ages - Gal Zaban

Gal Zaban hacks a game called "chicken invader" by reverse debugging through assembly of the game. This is a fun talk which shows the power of reverse debugging. IDA Pro is used for disassembling and debugging.

Compiled C++ Coding Standards - Valentin Galea

Valentin Galea presents how his team made their coding standard super convenient to use by making it compilable. It also prevents the standard from decaying over time and makes it democratic since changes in the standard are go through code review.

There are No Zero-cost Abstractions: Chandler Carruth

Chandler Caruth makes a case for why generic programming is not always free, through some examples in Google's code base which suffer from high compiling, linking time (e.g. protocol buffers while highly useful for representing complex data structure, moving them can be really expensive)

Key Takeaways

1. Moving abstractions from runtime to build time doesn't make them free.
2. unique ptr may not have the same runtime cost as raw pointer.

Type Punning in Modern C++: Timour Doumler

Timour Doumler talks about dangers of doing type punning using c-style casts, unions etc.

Type punning is undefined behaviour because of

- Aliasing Rules
- Lifetime of an object
- Alignment rulees
- Rules for value representation. (Related paper : Signed Ints are Two's Complement)

Key Takeaways:

1. Don't use C style casts.
2. Don't use union for type punning.
3. Use std::memcpy(std::variant)
4. Implicity Creation of Objects for low-level object manipulation

Small is beautiful: Tenchniques to minimise memory footprint - Steven Pigeon

Steven Pigeon who is professor at UQAR talks about his really interesting work on compressed pointers.

The amazing story of implementing Concepts in Clang - Saar Raz

Saar Raz talks about his super interesting story of how he implemented Concepts in Clang without any previous knowledge of the clang='s code base.

Smart Poiners - Arthur O'Dwyer

Arthur O'Dwyer gives a really good introduction into why and how of smart of pointers.

Better Code: Relationships - Sean Parent

Sean parent talks about art of designing relationships in your code. One particular example he gives is one has to think about safety and efficiency when managing resources and often one is compromised in favor of another (e.g. `std::move` is unsafe operation)

De-fragmenting C++ - Herb Sutter

This is the closing keynote. Herb Sutter talks about his future proposals for C++, on how to reduce cost of RTII and exceptions while also abstracting complexity.

What is C++ - Chandler Carruth, Titus Winters

This is the closing keynote of the back to basics track, which is wonderful discussion on design and pitfalls of C++ standard.