**()** codeplay<sup>®</sup> The future of C++ and heterogeneous programming Michael Wong **Codeplay Software** VP of Research and Development http:://wongmichael.com/about michael@codeplay.com

### VP of R&D of Codeplay

Chair of SYCL Heterogeneous Programming Language C++ Directions Group ISOCPP.org Director, VP http://isocpp.org/wiki/faq/wg21#michael-

wong Head of Delegation for C++ Standard for Canada

Chair of Programming Languages for Standards Council of Canada

Chair of WG21 SG19 Machine Learning

Chair of WG21 SG14 Games Dev/Low

Latency/Financial Trading/Embedded

Editor: C++ SG5 Transactional Memory

**Technical Specification** 

Editor: C++ SG1 Concurrency Technical

Specification

MISRA C++ and AUTOSAR

wongmichael.com/about

We build GPU compilers for

semiconductor companies

 Now working to make AI acceleration safe for automotive Ported TensorFlow to open standards using SYCL

Who am I?

Build LLVM-based compilers for accelerators

Releasing opensource, openstandards based AI acceleration tools: SYCL-BLAS, SYCL-ML, VisionCpp

Implement OpenCL and SYCL for accelerator processors

#### Acknowle dgement Disclaimer

Numerous people internal and external to the original C++/Khronos group, in industry and academia, have made contributions, influenced ideas, written part of this presentations, and offered feedbacks to form part of this talk.

Specifically, Paul Mckenney, Joe Hummel, Bjarne Stroustrup for some of the slides.

I even lifted this acknowledgement and disclaimer from some of them.

But I claim all credit for errors, and stupid mistakes. **These** are mine, all mine!

## Legal Disclaimer





THIS WORK REPRESENTS THE VIEW OF THE AUTHOR AND DOES NOT NECESSARILY REPRESENT THE VIEW OF CODEPLAY. OTHER COMPANY, PRODUCT, AND SERVICE NAMES MAY BE TRADEMARKS OR SERVICE MARKS OF OTHERS.

#### Codeplay - Connecting AI to Silicon



## 3 Act Play





1993

1994 1995 1996

1997

- Where is C++ Standard now?
- What is Parallelism in C++ 11, 14, 17, 20, 23?
- Is there a direction for C++?



International Organization Standardiza

• What gets me up every morning?



### C++11,14,17"No more Raw Food"

Don't use	Don't use raw numbers, do type-rich programming with UDL			
Don't declare	Don't declare, use auto whenever possible			
Don't use	Don't use raw NULL or (void *) 0, use nullptr			
Don't use	Don't use raw new and delete, use unique_ptr/shared_ptr			
Don't use	Don't use heap-allocated arrays, use std::vector and std::string, or the new VLA, then dynarray<>			
Don't use	Don't use functors, use lambdas			
Don't use	Don't use raw loops; use STL algorithms, ranged-based for loops, and lambdas			
Rule	Rule of Three? Rule of Zero or Rule of Five.			

### Parallelism "Use the right abstraction"

Abstraction	How is it supported
Cores	C++11/14/17 threads, async
HW threads	C++11/14/17 threads, async
Vectors	Parallelism TS2
Atomic, Fences, lockfree, futures, counters, transactions	C++11/14/17 atomics, Concurrency TS1, Transactional Memory TS1
Parallel Loops	Async, TBB:parallel_invoke, C++17 parallel algorithms, for_each
Heterogeneous offload, fpga	OpenCL, SYCL, HSA, OpenMP/ACC, Kokkos, Raja
Distributed	HPX, MPI, UPC++
Caches	C++17 false sharing support
Numa	Executors, Execution Context, Affinity
TLS	EALS
Exception handling in concurrent environment	EH reduction properties



### Act 1

- Where is C++ Standard now?
- What is Parallelism in C++ 11, 14, 17, 20, 23?
- Is there a direction for C++?



International Organization Standardiza

### C++ Standard ratification

Completed Issaquah, WA, USA (2014) Completed C++17 Kona, HI, USA (2017)

### C++ is more popular than ever



#### ISO C++ Standard



#### C++ Std Timeline/status https://isocpp.org/std/status





- No Concepts
- No Unified Call Syntax
- No Default Comparison
- No operator dot
- No Contracts
- No Modules
- No Transactional Memory



# So What is the best feature of C++17

- Parallel STL Algorithm
- Execution policy
- Thread of Execution
- Progress Guarantees

## Priorities for C++ 20

- Concepts (in the WP)
- Modules (offering proper modularity and dramatic compile-time improvements)
- Ranges (incl., some of the infinite sequence extensions to the TS )
- Networking which means
  - we need Executors in IS20
- Concepts in the standard library

## C++20 if time permits

- Contracts as specified in [GDR,2016], [GDR,2017].
- Coroutines [Nishanov,2017].
- Recommendation
  - Additions beyond that should be discouraged as time sinks and distractions.
  - Proposals for minor features should be given priority if and only if they support the priority items

## C++ 20 Language Features

- Most notably, the <u>Concepts</u> <u>Technical Specification has</u> <u>been merged into C++20!</u>
- <u>Template parameter lists for</u> <u>generic lambdas</u>. T
- Designated initializers.
- Lambda capture [=, \*this]
- A <u>VA OPT macro</u> to make variadic macros easier to use.
- <u>Default member initializers</u> for bitfields
- A tweak to C++17's constructor template argument deduction rules
- Fixing const-qualified pointers to members

- The most significant new feature voted in was <u>operator<=></u>,
- <u>Range-based for statements with</u> initializer.
- Lambdas is unevaluated contexts.
- Default constructible and assignable stateless lambdas.
- <u>Simplifying implicit lambda capture</u>.
- Fixing small functionality gaps in constraints.
- Deprecating the notion of "plain old data" (POD).
- Access checking on specializations.
- <u>const mismatch with defaulted copy</u> <u>constructor</u>.
- ADL and function templates that are not visible.
- Core issue 1581: when are constexpr member functions defined?

# More C++20 Language Features

- <u>Language support for empty</u> <u>objects</u>
- <u>Relaxing the structured</u> <u>bindings customization point</u> <u>finding rules</u>.
- <u>Structured bindings in</u> <u>accessible members</u>.
- <u>Allow pack expansion in</u> <u>lambda *init-capture*.</u>
- Symmetry for <=>
- Likely and unlikely attributes
- Down with typename!
- <u>Relaxing range-based for</u> <u>loop's customization point</u> <u>finding rules</u>

- <u>Support for contract-based</u> programming in C++20
- <u>Class types in non-type</u> <u>template parameters</u>.
- <u>Allowing virtual function</u> <u>calls in constant</u> <u>expressions</u>.
- <u>Prohibit aggregates with</u> <u>user-declared</u> <u>constructors</u>.
- Efficient sized deletion for variable-sized classes.

## More C++ 20 Language Features

- <u>Consistency improvements for <=> and</u> <u>other comparison operators</u>.
- <u>Conditionally explicit constructors</u>, a.k.a. explicit(bool).
- <u>Deprecate implicit capture of this via [=]</u>.
- <u>Integrating feature-test macros into the</u> <u>C++ working draft</u>.
- A tweak to the rules about <u>when certain</u> <u>errors related to a class being abstract are</u> <u>reported</u>.
- • A tweak to the <u>treatment of padding bits</u> during atomic compare-and-exchange operations.
- • Tweaks to the <u>VA OPT</u> preprocessor feature.
- Updating the reference to the Unicode standard.

- <u>Abbreviated function templates</u> (AFTs).
- <u>Improvements to return-type-</u> requirements.
- <u>Immediate functions</u>.
- std::is constant\_evaluated()
- try / catch blocks in constexpr functions.
- <u>Allowing dynamic cast and</u> polymorphic typeid in constant expressions.
- <u>Changing the active member of a</u> union inside constexpr
- char8 t: a type for UTF-8 characters and strings.
- Access control in contract conditions.
- <u>Revising the C++ memory model</u>.
- Weakening release sequences.
- Nested inline namespaces
- <u>Signed integers are two's complement</u>

## C++20 Library Features

- Support for <u>detecting</u> <u>endianness</u> <u>programmatically</u>
- <u>Repairing elementary</u> <u>string conversions</u> (also a Defect Report)
- Improvements to the integration of C++17 class template argument deduction into the standard library (also a Defect Report)
- Extending make shared to support arrays

- <u>Transformation trait remove\_cvref</u>
- <u>Treating unnecessary decay</u>
- Using nodiscard in the standard library
- <u>Make std::memory order a scoped</u>
  <u>enumeration</u>
- Synchronized buffered ostream
- A utility to <u>convert pointer-like</u> objects to raw pointers
- Add constexpr modifiers to functions in <algorithm> and <utility> headers.
- constexpr for std::complex
- Atomic shared ptr
- Floating-point atomics
- <u>De-pessimize legacy < numeric></u> algorithms with std::move
- <u>String prefix and suffix checking</u>, i.e. starts\_with() and ends\_with()

# More C++20 library Features

- <u>calendar and timezone</u> <u>library</u>.
- <u>std::span</u>
- <version> header
- Tweak on <u>how unordered</u> <u>containers are compared</u>
- <u>String::reserve() should not</u> <u>shrink</u>
- User specializations of function templates in namespace std
- <u>Manipulators for C++</u> synchronized buffer ostream
- <u>constexpr iterator</u> <u>requirements</u>

- The most notable addition at this meeting was **standard library Concepts**.
- <u>atomic ref</u>
- <u>Bit-casting object representations</u>
- <u>Standard library specification in a</u> <u>Concepts and Contracts world</u>
- <u>Checking for the existence of an element</u> in associative containers
- Add shift() to <algorithm>
- Implicit conversion traits and utility functions
- Integral power-of-2 operations
- The identity metafunction
- Improving the return value of erase()-like algorithms
- <u>constexpr</u> comparison operators for std::array
- <u>constexpr</u> for swap and related functions
- <u>fpos requirements</u>
- <u>Eradicating unnecessarily explicit default</u> constructors
- Removing some facilities that were deprecated in C++17 or earlier

### More C++20 Library Features

- The most notable addition at this meeting was merging the <u>Ranges TS</u> into C++20!
- Fixing operator>>(basic\_istream&, CharT\*).
- variant and optional should propagate copy/move triviality.
- visit<R>: explicit return type for visit.
- <chrono> zero(), min(), and max() should be noexcept.
- <u>constexpr in std::pointer\_traits.</u>
- Miscellaneous constexpr bits.
- <u>unwrap ref decay and unwrap reference</u>
- <u>reference\_wrapper for incomplete types</u>
- <u>A sane variant converting constructor</u>
- <u>std::function move constructor should be</u> <u>noexcept</u>

- <u>std::assume\_aligned</u>
- <u>Smart pointer creation with default</u> <u>initialization</u>
- <u>Improving completeness requirements for</u> <u>type traits</u>)
- <u>Remove CommonReference requirement from</u> <u>StrictWeakOrdering (a.k.a fixing relations)</u>
- <u>Utility functions to implement uses-allocator</u> <u>construction</u>
- Should span be Regular?
- <u>Make stateful allocator propagation more</u> <u>consistent for operator+(basic\_string))</u>
- Simplified partial function application
- <u>Heterogeneous lookup for unordered</u> <u>containers</u>
- Adopt consistent container erasure from Library Fundamentals v2

#### Pre-C++11 projects

ISO number	Name	Status	What is it?	C++17?
ISO/IEC TR 18015:2006	Technical Report on C++ Performance	Published 2006 ( <u>ISO store</u> ) Draft: <u>TR18015</u> (2006-02-15)	C++ Performance report	No
ISO/IEC TR 19768:2007	Technical Report on C++ Library Extensions	Published 2007-11-15 ( <u>ISO store</u> ) Draft: <u>n1745</u> (2005-01-17) TR 29124 split off, the rest merged into C++11	Has 14 Boost libraries, 13 of which was added to C++11.	N/A (mostly already included into C++11)
ISO/IEC TR 29124:2010	Extensions to the C++ Library to support mathematical special functions	Published 2010-09-03 ( <u>ISO Store</u> ) Final draft: <u>n3060</u> (2010-03-06). Under consideration to merge into C++17 by <u>p0226</u> (2016- 02-10)	Really, ORDINARY math today with a Boost and Dinkumware Implementation	YES
ISO/IEC TR 24733:2011	Extensions for the programming language C++ to support decimal floating-point arithmetic	Published 2011-10-25 ( <u>ISO Store</u> ) Draft: <u>n2849</u> (2009-03-06) May be superseded by a future Decimal TS or merged into C++ by <u>n3871</u>	Decimal Floating Point decimal32 decimal64 decimal128	No. Ongoing work in SG6

#### Status after Nov SAN C++ Meeting

ISO NUMBER	NAME	STATUS	LINKS	C++20?
ISO/IEC TS 19841:2015	Transactional Memory TS	Published 2015-09-16, ( <u>ISO Store</u> ). Final draft: <u>n4514</u> (2015-05-08)	Composable lock-free programming that scales	No. Already in GCC 6 release and waiting for subsequent usage experience.
ISO/IEC TS 19217:2015	C++ Extensions for Concepts	Published 2015-11-13. ( <u>ISO Store</u> ). Final draft: <u>n4553</u> (2015-10-02) Current draft: <u>p0734r0</u> (2017-07- 14) Merged into C++20 (with modifications).	Constrained templates	Merged into C++20, <b>including</b> (now) abbreviated function templates!
	Executors		Abstraction for where/how code runs in a concurrent context	Lite form headed for C++20, rest aiming for C++23
	Coroutines TS		Resumable functions, based on Microsoft's await design	Published! C++20 merge uncertain
	Reflection TS		Static code reflection mechanisms	PDTS ballot underway; publication expected in early 2019

Concepts: compromised design for Abbreviated Function Template

void f(Concept auto x); Concept auto f(Concept auto x);

#### Status after Nov SAN C++ Meeting

ISO number	Name	Status	What is it?	C++20?
ISO/IEC TS 19571:2016	C++ Extensions for Concurrency	Published 2016-01-19. ( <u>ISO Store</u> ) Final draft: <u>p0159r0</u> (2015-10-22)	improvements to future, latches and barriers, atomic smart pointers	Latches, atomic <shared_ptr<t>&gt; merged into C++20. Already in Visual Studio release and Anthony Williams Just Threads! and waiting for subsequent usage experience.</shared_ptr<t>
ISO/IEC TS 19568:2017	C++ Extensions for Library Fundamentals, Version 2	Published 2017-03-30. ( <u>ISO Store</u> ) Draft: <u>n4617</u> (2016-11-28)	source code information capture and various utilities	Published! Parts of it merged into C++17
ISO/IEC DTS 21425:2017	Ranges TS	Published 2017-12-05. ( <u>ISO Store</u> ) Draft: <u>n4685</u> (2017-07-31)	Range-based algorithms and views	Merged in C++20
ISO/IEC TS 19216:2018	Networking TS	Published 2018-04-24. ( <u>ISO Store</u> ) Draft <u>n4734</u> (2017-04-04). Latest draft: <u>n4771</u> (2018-10-08)	Sockets library based on Boost.ASIO	Published. But may not be added to C++20.
ISO/IEC TS 21544:2018	Modules V1	Published 2018-05-16. ( <u>ISO Store</u> ) Final Draft <u>n4720</u> (2018-01-29)	A component system to supersede the textual header file inclusion model	Published as a TS
	Modules V2		Improvements to Modules v1, including a better transition path	On track to be merged into C++20

#### Status after Nov SAN C++ Meeting

ISO number	Name	Status	What is it?	C++20?
ISO/IEC DTS 19568:xxxx	Numerics TS	Early development. Draft p0101 (2015-09-27)	Various numerical facilities	Under active development
ISO/IEC DTS 19571:xxxx	Concurrency TS 2	Early development	Exploring , lock-free, hazard pointers, RCU, atomic views, concurrent data structures	Under active development. Possible new clause
ISO/IEC TS 19570:2018	Parallelism TS 2	Published 2018-11-15. ( <u>ISO</u> <u>Store</u> ). Draft: <u>n4773</u> (2018- 10-08)	task blocks, progress guarantees, SIMD <t>, vec, no_vec loop based execution policy</t>	Published. Headed into C++20
ISO/IEC DTS 19841:xxxx	Transactional Memory TS 2	Early development	Exploring on_commit, in_transaction. Lambda- based executor model.	Under active development.
ISO/IEC DTS 19568:xxxx	Graphics TS	Early development. Draft p0267r8 (2018-06-26)	2D drawing API using Cairo interface, adding stateless interfacec	Restarted after being shutdown.
ISO/IEC DTS 19568:xxxx	Library Fundamental V3	Initial draft, early development	Maybe mdspan and expected <t></t>	Under development



### Act 2

- Where is C++ Standard now?
- What is Parallelism in C++ 11, 14, 17, 20, 23?
- Is there a direction for C++?

# The Parallel and concurrency planets of C++ today



#### So Why do we need to standardize concurrency?

#### Can help existing advanced abstractions •TBB, PPL, Cilk,

#### **Reflects the real world**

Multi-core processorsSolutions for very large problemsInternet programming

#### Standardize existing practice

- •C++ threads=OS threads
- shared memory
- •Loosely based on POSIX, Boost thread
- Does not replace other specifications
  MPI, OpenMP, UPC, autoparallelization, many others



#### Concurrency Language and Library

Core: what does it mean to share memory and how it affects variables

#### • TLS

- Static duration variable initialization/destruction
- Memory model
- Atomic types and operations
- Lock-free programing
- Fences
- Dependence based Ordering

 Library: threads and mutexes

- How to create/synchronize/terminate threads,
- Thread , mutex , locks
- RAII for locking, type safe
- propagate exceptions
- A few advanced abstraction
- Async(), promises and futures
- parallel STL

# What we got in C++



•System level interoperability

•C++ shares threads with other languages

# What we are still trying to get

#### •All the nifty, higher parallel abstractions

- TM, atomic<shared<T>>, queues and counters
- SIMD, Task Blocks, coroutines, networking
- Distributed and Heterogeneous programming
- Reactive programming

#### •Complete Compatibility between C and C++

•Total isolation from programmer mistakes



#### Coverage before C++11 (C++98)

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous (GPUs, accelerators, FPGA, embedded AI processors)
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
examples	GUI,background printing, disk/net access	trees, quicksorts, compilation	locked data(99%), lock-free libraries (wizards), atomics (experts)	Pipelines, reactive programming, offload,, target, dispatch
key metrics	responsiveness	throughput, many core scalability	race free, lock free	Independent forward progress,, load-shared
requirement	isolation, messages	low overhead	composability	Distributed, heterogeneous
today's abstractions	POSIX threads, win32 threads, OpenCL, vendor intrinsic	openmp, TBB, PPL, OpenCL, vendor intrinsic	locks, lock hierarchies, vendor atomic instructions, vendor intrinsic	OpenCL, CUDA
### Coverage after C++11

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous (GPUs, accelerators, FPGA, embedded AI processors)
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
examples	GUI,background printing, disk/net access	trees, quicksorts, compilation	locked data(99%), lock-free libraries (wizards), atomics (experts)	Pipelines, reactive programming, offload,, target, dispatch
key metrics	responsiveness	throughput, many core scalability	race free, lock free	Independent forward progress,, load-shared
requirement	isolation, messages	low overhead	composability	Distributed, heterogeneous
today's abstractions	C++11: thread,lambda function, TLS	C++11: Async, packaged tasks, promises, futures, atomics	C++11: locks, memory model, mutex, condition variable, atomics, static init/term	C++11: lambda

### Coverage after C++14

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
examples	GUI,background printing, disk/net access	trees, quicksorts, compilation	locked data(99%), lock-free libraries (wizards), atomics (experts)	Pipelines, reactive programming, offload,, target, dispatch
key metrics	responsiveness	throughput, many core scalability	race free, lock free	Independent forward progress,, load-shared
requirement	isolation, messages	low overhead	composability	Distributed, heterogeneous
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda	C++11: Async, packaged tasks, promises, futures, atomics,	C++11: locks, memory model, mutex, condition variable, atomics, static init/term,	C++11: lambda C++14: none
			C++ 14: shared_lock/shared_timed_ mutex, OOTA, atomic_signal_fence,	

### Coverage after C++17

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous (GPUs, accelerators, FPGA, embedded AI processors)
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParalleISTL, control false sharing	C++11: locks, memory model, mutex, condition variable, atomics, static init/term, C++ 14: shared_lock/shared_timed_ mutex, OOTA, atomic_signal_fence, C++ 17: scoped _lock, shared_mutex, ordering of memory models, progress guarantees, TOE, execution policies	C++17: , progress guarantees, TOE, execution policies

### Coverage aiming for C++20

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous/Distributed
today's abstractions	C++11: thread,lambda function, TLS, async C++ 20: Executors Lite Jthreads +interrupt _token	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing C++ 20: Is_ready(), make_ready_future() Task blocks simd <t>, Vec execution policy, Algorithm un- sequenced policy Executors Lite, mdspan</t>	C++11: locks, memory model, mutex, condition variable, atomics, static init/term, C++ 14: shared_lock/shared_timed_ mutex, OOTA, atomic_signal_fence, C++ 17: scoped _lock, shared_mutex, ordering of memory models, progress guarantees, TOE, execution policies C++20: atomic_ref, Latches and barriers atomic <shared_ptr> Atomics &amp; padding bits Simplified atomic init Atomic C/C++ compatibility Semaphores and waiting Fixed gaps in memory model , Improved atomic flags, Repair memory model</shared_ptr>	C++17: , progress guarantees, TOE, execution policies C++20: atomic_ref, mdspan, executors Lite



#### Unified interface for execution



# **Current Progress of Executors**

- An *instruction stream* is the function you want to execute
- An *executor* is an interface that describes where and when to run an *instruction stream*
- An *executor* has one or more *execute functions*
- An *execute function* executes an *instruction stream* on light weight *execution agents* such as threads, SIMD units or GPU threads



# **Current Progress of Executors**

- An *execution platform* is a target architecture such as linux x86
- An *execution resource* is the hardware abstraction that is executing the work such as a thread pool
- An execution context manages the light weight execution agents of an execution resource during the execution



### Coverage beyond C++20: C++23

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous/DIstributed
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda C++ 20: Executors Lite Jthreads +interrupt _token C++23: coroutines, networking, asynchronous algorithm, reactive programming, EALS, async2	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParalleISTL, control false sharing C++ 20: Is_ready(), make_ready_future() Task blocks simd <t>, Vec execution policy, Algorithm un- sequenced policy Executors Lite, mdspan</t>	C++11: C++ 14: C++ 17: C++20: atomic_ref, Latches and barriers atomic <shared_ptr> Atomics &amp; padding bits Simplified atomic init Atomic C/C++ compatibility Semaphores and waiting Fixed gaps in memory model , Improved atomic flags , Repair memory model</shared_ptr>	C++17: , progress guarantees, TOE, execution policies C++20: atomic_ref, mdspan, executors Lite C++23: affinity, pipelines, EALS, freestanding/embedded support well specified, mapreduce, ML/AI, reactive programming
		C++23: new futures, concurrent vector, unordered associative containers, two-way executors with lazy sender- receiver models, concurrent exception handling,	C++23: hazard_pointers, rcu/snapshot, concurrent queues, counters, upgrade lock, TM lite, more lock- free data structures, asymmetric fences	



```
{
  auto exec = execution::execution_context{execRes}.executor();
  auto affExec = execution::require(exec, execution::bulk,
    execution::bulk_execution_affinity.compact);
  affExec.bulk_execute([](std::size_t i, shared s) {
    func(i);
    }, 8, sharedFactory);
}
```

# Act 3

- Where is C++ Standard now?
- What is Parallelism in C++ 11, 14, 17, 20, 23?
- Is there a direction for C++?



Directions for ISO C++

C++

Directions

Group:

P0939

DWG

P0939r0

Doc. no.: P0939r0 Date: 2018-02-10 Programming Language C++ Audience: All WG21 Reply to: Bjarne Stroustrup (<u>bs@ms.com</u>)

#### Direction for ISO C++

B. Dawes, H. Hinnant, B. Stroustrup, D. Vandevoorde, M. Wong

#### **Revision History**

• This is the initial version.

#### Main sections

- <u>History</u>
- Long-term Aims (decades)
- Medium-term Aims (3-10 years)
- Priorities for C++20
- Process Issues
- The C++ Programmer's Bill of Rights

I have a big idea for a big change  Change gradually building on previous work

• OR

 Provide better alternative to existing feature Many cooks (photos by Bjarne Stroustrup)

0

# I have a secret to tell you



Direction Group created as Pri response to Call to C++ Action of

Operating Principles for C++ by Heads of Delegation

C++ in danger of losing coherency due to proposals with differ and contradictory design philosophies The Direction Group direction@lists.isocpp.org

We try to represent USERS: the Interest of the larger C++ community







## WG 21 Direction Group



# What is C++







C++ is a language for defining and using lightweight abstractions C++ supports building resource constrained applications and software infrastructure C++ support large-scale software development

# How do we want C++ to develop?



*Improve support for large -scale dependable software* 



*Improve support for high-level concurrency models* 



Simplify language use



Address major sources of dissatisfaction



Address major sources of error

C++ rests on two pillars • A direct map to hardware (initially from C)

• Zero-overhead abstraction in production code (initially from Simula, where it wasn't zero-overhead)

# Strengthen two pillars

Better support for modern hardware (e.g., concurrency, GPUs, FPGAs, NUMA architectures, distributed systems, new memory systems)

More expressive, simpler, and safer abstraction mechanisms (without added overhead)

# 4.3 Concrete Suggestions

- Pattern matching
- Exception and error returns
- Static reflection
- Modern networking
- Modern hardware:
- We need better support for modern hardware, such as executors/execution
- context, affinity support in C++ leading to **heterogeneous/distributed** computing support,
- SIMD/task blocks, more concurrency data structures, improved atomics/memory model/lock-
- *free data structures support. The challenge is to turn this (incomplete) laundry list into a*
- coherent set of facilities and to introduce them in a manner that leaves each new standard with
- a coherent subset of our ideal.
- Simple graphics and interaction
- Anything from the Priorities for C++20 that didn't make C++20

### Modern hardware

• We need better support for modern hardware, such as executors/execution context, affinity support in C++ leading to heterogeneous/distributed computing support, ...

# What have we achieved so far?

	Depends on	Current target (estimated, could slip)
Concepts		C++20 (adopted, including convenience syntax)
Contracts		C++20 (adopted)
Ranges		C++20 (adopted)
Coroutines		C+>xX
Modules		C++20
Reflection		TS in C++20 timeframe, IS in C++23
Executors		Lite in C++20 timeframe, Full in C++23
Networking	Executors, and possibly Coroutines	C++23
future.then, async2	Executors	

# Use the Proper Abstraction with C++

Abstraction	How is it supported
Cores	C++11/14/17 threads, async
HW threads	C++11/14/17 threads, async
Vectors	Parallelism TS2->C++20
Atomic, Fences, lockfree, futures, counters, transactions	C++11/14/17 atomics, Concurrency TS1->C++20, Transactional Memory TS1
Parallel Loops	Async, TBB:parallel_invoke, C++17 parallel algorithms, for_each
Heterogeneous offload, fpga	OpenCL, SYCL, HSA, OpenMP/ACC, Kokkos, Raja P0796 on affinity
Distributed	HPX, MPI, UPC++ P0796 on affinity
Caches	C++17 false sharing support
Numa	Executors, Execution Context, Affinity, P0443->Executor TS or IS20
TLS	EALS, P0772
Exception handling in concurrent environment	EH reduction properties P0797

# If you have to remember 3 things



# .Oh, and one more thing

### C++23: Heterogeneous

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous/Distributed
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda C++ 20: Executors Lite Jthreads +interrupt _token C++23: coroutines, networking, asynchronous algorithm, reactive programming, EALS, async2	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing C++ 20: Is_ready(), make_ready_future() Task blocks simd <t>, Vec execution policy, Algorithm un-sequenced policy Executors Lite, mdspan C++23: new futures, concurrent vector, unordered associative containers, two- way executors with lazy sender-receiver models, concurrent exception handling,</t>	C++11: C++ 14: C++ 17: C++ 20: atomic_ref, Latches and barriers atomic <shared_ptr> Atomics &amp; padding bits Simplified atomic init Atomic C/C++ compatibility Semaphores and waiting Fixed gaps in memory model , Improved atomic flags , Repair memory model C++23: hazard_pointers, rcu/snapshot, concurrent queues, counters, upgrade lock, TM lite, more lock-free data structures, asymmetric fences</shared_ptr>	C++17: , progress guarantees, TOE, execution policies C++20: atomic_ref, mdspan, executors Lite C++23: affinity, pipelines, EALS, freestanding/embedded support well specified, mapreduce, ML/AI, reactive programming



Intel Core i7 7th generation

size\_t nElems = 1000u; std::vector<float> nums(nElems);

std::fill\_n(std::begin(v1), nElems, 1);

std::for\_each(std::begin(v), std::end(v),
 [=](float f) { f \* f + f });
Traditional for each uses only one core,
 rest of the die is unutilized!



Intel Core i7 7th generation

size\_t nElems = 1000u; std::vector<float> nums(nElems);

#### Workload is distributed across cores!

(mileage may vary, implementation-specific behaviour)



Intel Core i7 7th generation

size\_t nElems = 1000u; std::vector<float> nums(nElems);

std::for\_each(std::execution\_policy::par,

What about this part?

std::begin(v), std::end(v),
[=](float f) { f \* f + f });

#### Workload is distributed across cores!

(mileage may vary, implementation-specific behaviour)



Intel Core i7 7th generation



size\_t nElems = 1000u; std::vector<float> nums(nElems);

std::fill\_n(**sycl\_policy**, std::begin(v1), nElems, 1);

(mileage may vary, implementation-specific behaviour)





### Demo Results - Running std::sort (Running on Intel i7 6600 CPU & Intel HD Graphics 520)

size	2^16	2^17	2^18	2^19
std::seq	0.27031s	0.620068s	0.669628s	1.48918s
std::par	0.259486s	0.478032s	0.444422s	1.83599s
std::unseq	0.24258s	0.413909s	0.456224s	1.01958s
sycl_execution_policy	0.273724s	0.269804s	0.277747s	0.399634s

### SYCL Ecosystem

- ComputeCpp -<u>https://codeplay.com/products/computesuite/computecpp</u> triSYCL - <u>https://github.com/triSYCL/triSYCL</u>
- SYCL http://sycl.tech
- SYCL ParalleISTL <u>https://github.com/KhronosGroup/SyclParalleISTL</u>

- VisionCpp <u>https://github.com/codeplaysoftware/visioncpp</u>
   SYCL-BLAS <u>https://github.com/codeplaysoftware/sycl-blas</u>
   TensorFlow-SYCL <u>https://github.com/codeplaysoftware/tensorflow</u>
   Eigen <u>http://eigen.tuxfamily.org</u>

### **Eigen Linear Algebra Library**

SYCL backend in mainline Focused on Tensor support, providing support for machine learning/CNNs Equivalent coverage to CUDA Working on optimization for various hardware architectures (CPU, desktop and mobile GPUs) https://bitbucket.org/eigen/eigen/


### TensorFlow

SYCL backend support for all major CNN operations Complete coverage for major image recognition networks GoogLeNet, Inception-v2, Inception-v3, ResNet, ....

Ongoing work to reach 100% operator coverage and optimization for various hardware architectures (CPU, desktop and mobile GPUs)

https://github.com/tensorflow/tensorflow



TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

### SYCL Ecosystem

- Single-source heterogeneous programming using STANDARD C++
  - Use C++ templates and lambda functions for host & device code
  - Layered over OpenCL
- Fast and powerful path for bring C++ apps and libraries to OpenCL
  - C++ Kernel Fusion better performance on complex software than hand-coding
  - Halide, Eigen, Boost.Compute, SYCLBLAS, SYCL Eigen, SYCL TensorFlow, SYCL GTX
  - triSYCL, ComputeCpp, VisionCpp, ComputeCpp SDK ...
- More information at <u>http://sycl.tech</u>

#### **Developer Choice**

The development of the two specifications are aligned so code can be easily shared between the two approaches

C++ Kernel Language Low Level Control 'GPGPU'-style separation of device-side kernel source code and host code



Single-source C++ Programmer Familiarity Approach also taken by C++ AMP and OpenMP





Codep	lay									
Standards bodies	Research		Open source			Presentations			Company	
<ul> <li>HSA Foundation: Chair of software group, spec editor of runtime and debugging</li> <li>Khronos: Chair &amp; spec editor of SYCL. Contributors to OpenCL, Safety Critical, Vulkan</li> <li>ISO C++: Chair of Low Latency, Embedded WG; Editor of SG1 Concurrency TS</li> <li>EEMBC: members</li> </ul>	<ul> <li>Members of EU research consortiums: PEPPHER, LPGPU, LPGPU2, CARP</li> <li>Sponsorship of PhDs and EngDs for heterogeneous programming: HSA, FPGAs, ray-tracing</li> <li>Collaborations with academics</li> <li>Members of HiPEAC</li> </ul>		<ul> <li>HSA LLDB Debugger</li> <li>SPIR-V tools</li> <li>RenderScript debugger in AOSP</li> <li>LLDB for Qualcomm Hexagon</li> <li>TensorFlow for OpenCL</li> <li>C++ 17 Parallel STL for SYCL</li> <li>VisionCpp: C++ performance-portable programming model for vision</li> </ul>			<ul> <li>Building an LLVM back-end</li> <li>Creating an SPMD Vectorizer for OpenCL with LLVM</li> <li>Challenges of Mixed-Width Vector Code Gen &amp; Scheduling in LLVM</li> <li>C++ on Accelerators: Supporting Single- Source SYCL and HSA</li> <li>LLDB Tutorial: Adding debugger support for your target</li> </ul>			<ul> <li>Based in Edinburgh, Scotland</li> <li>57 staff, mostly engineering</li> <li>License and customize technologies for semiconductor companies</li> <li>ComputeAorta and ComputeCpp: implementations of OpenCL, Vulkan and SYCL</li> <li>15+ years of experience in heterogeneous systems tools</li> </ul>	
VectorC for x86 Our VectorC technology was chosen and actively used for Computer Vision First showing of VectorC{VU}		Sieve C++ Pro System relea: Aimed at helping to parallelise C++ evaluated by nur researchers	ogramming ised g developers + code, imerous			LLDB Machine Interface Driver released			Open-Source HSA Debugger release Releases partial OpenCL support (via SYCL) for Eigen Tensors to power Tensorflow	
Delivered VectorC(VU) to the National Center for Supercomputing VectorC(EE) released An optimising C/C++ compiler for PlayStation®2 Emotion Engine (MIPS)	Ageia chooses Codeplay for PhysX Codeplay is chosen by Ageia to provide a compler for the PhysX processor. Codeplay joins the Khronos Group	Offload relea: Sony PlaySta OffloadCL tec developed Codeplay joir PEPPHER pro	sed for tion@3 chnology is the ject	New R&D Division Codeplay forms a new R&D division to develop innovative new standards and products Becomes specification editor of the SYCL standard	Code proj Code tech acce on O	eplay joins the CARP ect eplay shows nology to elerate Renderscript JpenCL using SPIR	Chair of HSA Syste Runtime working g Development of to supporting the Vul API	m roup ols kan	ComputeAorta 1.0 release ComputeCpp Community Edition beta release First public edition of Codeplay's SVCL technology	
2001 - 2003	2005 - 2006	2007 - 20'	11	2013		2014	2015		2016	

Codeplay build the software platforms that deliver massive performance

## What our ComputeCpp users say

#### TensorFlow

about us

Benoit Steiner – Google TensorFlow

enaineer

"We at Google have been working closely with Luke and his Codeplay colleagues on this project for almost 12 months now. Codeplay's contribution to this effort has been tremendous, so we felt that we should let them take the lead when it comes down to communicating updates related to OpenCL. ... we are planning to merge the work that has been done so far... we want to put together a comprehensive test infrastructure"



"We work with royalty-free SYCL because it is hardware vendor agnostic, singlesource C++ programming model without platform specific keywords. This will allow us to easily work with any heterogeneous processor solutions using OpenCL to develop our complex algorithms and ensure future compatibility" "My team and I are working with Codeplay's ComputeCpp for almost a year now and they have resolved every issue in a timely manner, while demonstrating that this technology can work with the most complex C++ template code. I am happy to say that the combination of Codeplay's SYCL implementation with our HPX runtime system has turned out to be a very capable basis for Building a Heterogeneous Computing Model for the C++ Standard using high-level abstractions."

Hartmut Kaiser - HPX

WIGNER Research Centre for Physics



It was a great pleasure this week for us, that Codeplay released the ComputeCpp project for the wider audience. We've been waiting for this moment and keeping our colleagues and students in constant rally and excitement. We'd like to build on this opportunity to increase the awareness of this technology by providing sample codes and talks to potential users. We're going to give a lecture series on modern scientific programming providing field specific examples."

# Further information

- OpenCL <u>https://www.khronos.org/opencl/</u>
- OpenVX <u>https://www.khronos.org/openvx/</u>
- HSA <u>http://www.hsafoundation.com/</u>
- SYCL <u>http://sycl.tech</u>
- OpenCV <u>http://opencv.org/</u>
- Halide <u>http://halide-lang.org/</u>
- VisionCpp <u>https://github.com/codeplaysoftware/visioncpp</u>



## **Community Edition**

Available now for free!

Visit: computecpp.codeplay.com



- Open source SYCL projects:
  - ComputeCpp SDK Collection of sample code and integration tools
  - SYCL ParallelSTL SYCL based implementation of the parallel algorithms
  - VisionCpp Compile-time embedded DSL for image processing
  - Eigen C++ Template Library Compile-time library for machine learning

All of this and more at: <u>http://sycl.tech</u>



## Questions ?







@codeplaysoft

/codeplaysoft

codeplay.com