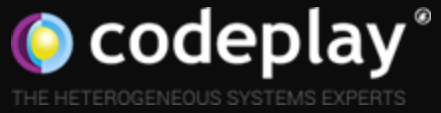


ISO C++ Parallelism, Concurrency and Heterogeneous futures

Gordon Brown & Michael Wong

CppCon 2020 – Sep 2020

- Learning objectives:
  - Learn about the Quiet Revolution in ISO C++ for heterogeneous computing
  - Learn about ISO C++ 11, 14, 17, 20, and future parallelism
  - Learn about ISOC C++ directions for future Heterogeneous support

OpenCL

OpenMP

SYCL

CUDA

The Quiet Revolution

Kokkos

HPX

Raja

Boost.Compute

Parallel/concurrency 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

Parallel/concurrency 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, Async	C++11: Async, packaged tasks, promises, futures, atomics	C++11: locks, memory model, mutex, condition variable, atomics, static init/term	C++11: lambda

Parallel/concurrency 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++ 14: shared_lock/shared_timed_mutex, OOTA, atomic_signal_fence,	C++11: lambda C++14: none

Parallel/concurrency 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: ParallelSTL, 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++11: lambda C++14: generic lambda C++17: progress guarantees, TOE, execution policies

Parallel/concurrency 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: Jthreads +interrupt _token, coroutines	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing C++ 20: Is_ready(), make_ready_future(), simd<T>, Vec execution policy, Algorithm un-sequenced policy, span	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 & padding bits Simplified atomic init Atomic C/C++ compatibility Semaphores and waiting Fixed gaps in memory model , Improved atomic flags, Repair memory model	C++11: lambda C++14: generic lambda C++17: , progress guarantees, TOE, execution policies C++20: atomic_ref,, span

Parallel/Concurrency 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: Jthreads +interrupt _token C++23: networking, asynchronous algorithm, reactive programming, EALS, async2, executors	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing C++ 20: Is_ready(), make_ready_future(), Vec execution policy, Algorithm un-sequenced policy span C++23: SMD<T>,new futures, concurrent vector,task blocks, unordered associative containers, two-way executors with lazy sender-receiver models, concurrent exception handling, executors, mdspan	C++11: … C++ 14: … C++ 17: … C++20: atomic_ref, Latches and barriers atomic<shared_ptr> Atomics & 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	C++17: , progress guarantees, TOE, execution policies C++20: atomic_ref, mdspan, C++23: SIMD<T>,affinity, pipelines, EALS, freestanding/embedded support well specified, mapreduce, ML/AI, reactive programming executors, mdspan

C++23: continue C++20

- Library support for coroutines
- Further Conceptifying Standard Library
- Further Range improvements (e.g., application of ranges to parallel algorithms and operations on containers and integration with coroutines)
- A modular standard library

After C++20

- Much more libraries
  - Audio
  - Linear Algebra
  - Graph data structures
  - Tree Data structures
  - Task Graphs
  - Differentiation
  - Reflection
  - Light-weight transactional locks
  - A new future and/or a new async
  - Statistics Library
  - Array style programming through mdspan

- Machine learning support
- Executors
- Networking
- Pattern Matching
- Better support for C++Tooling ecosystem
- Further support for heterogeneous programming
- Graphics
- Better definition of freestanding
- Education dependency curriculum

After C++23

- Reflection
- Pattern matching
- C++ ecosystem
- What about Contracts?

What have we achieved so far for C++20?

	Depends on	Current target (estimated, could slip)
Concepts		C++20 (adopted, including convenience syntax)
Contracts		C++20 (adopted)
Ranges		C++20 (adopted)
Coroutines		C++20
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++ 20

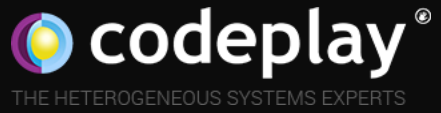
Abstraction	How is it supported
Cores	C++11/14/17 threads, async
HW threads	C++11/14/17 threads, async
Vectors	Parallelism TS2-
Atomic, Fences, lockfree, futures, counters, transactions	C++11/14/17 atomics, Concurrency TS1->C++20, Transactional Memory TS1
Parallel Loops	Async, TBB:parallel_invoke, C++17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

Key takeaways

A quiet revolution towards heterogenous programming is taking place in ISO C++, though it will take 5-10 years for full support

ISO C++ 11,14,17,20 contains increasing support for higher abstraction for parallel and concurrency constructs

Even C++11 has elements that supported heteoregenous programming thus enabling SYCL



Questions?