Misra Parallelism Safety-critical Guidelines for C++11, 17, Then C++20, 23

ANDREAS WEIS, MICHAEL WONG & ILYA BURYLOV
Staff Engineer at Woven Planet

Andreas Weis (he/him)

@DerGhulbus

Co-organizer of the Munich C++ User Group (MUC++)

Member of WG21 (ISO C++) and MISRA C++

Working on the Runtime framework for the Arene platform at Woven Planet
Principle Engineer
at Intel

Ilya Burylov

An architect of C++ software solutions for autonomous driving market in Intel

Contribution into functional safety MISRA standard

Contribution into WG21 in threading, vectorization and numerics.

Contribution into SYCL
Michael Wong

Distinguished Engineer

- Chair of SYCL Heterogeneous Programming Language
- ISO C++ Directions Group past Chair
- Past CEO OpenMP
- ISOCPP.org Director, VP
  http://isocpp.org/wiki/faq/wg21#michael-wong
  michael@codeplay.com
  fraggamuffin@gmail.com
- 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
- Chair of Standards Council Canada TC22/SC32 Electrical and electronic components (SOTIF)
- Chair of UL4600 Object Tracking
- RISC-V Datacenter/Cloud Computing Chair
- http://wongmichael.com/about
- C++11 book in Chinese:
  https://www.amazon.cn/dp/B00ETOV2OQ

We build GPU compilers for some of the most powerful supercomputers in the world
Acknowledgement and 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.

Images belong to their respective copyrights.

But I claim all credit for errors, and stupid mistakes. These are mine, all mine! You can’t have them.
Agenda

1. Current status of C++ safety: MISRA and C++ CG
2. Parallel Safety rules
3. Automotive Safety case
Safety Critical API Evolution

Industry Need
for CPU/GPU Acceleration APIs designed to ease system safety certification

minimize API surface area, reduce ambiguity, UB, increase determinism

New Generation Safety Critical APIs for Graphics, Compute and Display
Many Safety Critical APIs

- **Misra**: checkable rules only
- **Autosar C++ Guidelines**: a mix of meta guidelines and checkable rules
- **High Integrity C++**: for static checkers
- **WG23 Programming Vulnerabilities**: for team leads
- **C++ Core Guidelines**: a mix
- **C++ Study Group 12 Vulnerabilities**: for standards
- **C Safe and Secure Study Group**: for standards
- **Carnegie Mellon Cert C and C++**: a mix
- **Joint Strike Fighter ++**: checkable rules
- **Common Weakness Enumeration**: a mix

- **Khronos Safety Critical Advisory Forum**
- **OpenCL/SYCL Safety Critical**
- **Vulkan Safety Critical**
- **JTC1/SC42 Machine Learning WG3 Trustworthiness**
- **ITC22/SC32 SOTIF** WG8 SOTIF, WG13, WG14
- **SAE ORAD**
- **UL4600**
- **RISC-V Safety/Security**
Which one to choose and what is the difference?

- Safe but not C++11/14/17/20
  - Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program, 2005
    - With the help of Bjarne Stroustrup
    - Continues to be the reference despite its age
    - For automated static analysis tools
    - Aimed for embedded domains

- C++11/14/17/20 but not safe
  - High Integrity C++ Coding Standard Version 4.0, Programming Research Ltd, 2013
    - Some parallelism
  - Software Engineering Institute CERT C/C++ Coding Standard, Software Engineering Institute Division at Carnegie Mellon University, 2016
    - Most recent effort based on C 11 and C++ 14
    - Most recent effort based on C++17 + 20
    - An excellent style guide for greater elegance, and safety/performance
    - No specific domains, also for static analysis and guidance
  - WG23 Vulnerabilities ISO for C, C++, Ada, Fortran, …
    - Guidelines for teamleads
    - Reviewed with each ISO C, C++, Ada, Fortran help
Comparing coding standards

<table>
<thead>
<tr>
<th>Coding Standard</th>
<th>C++ Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autosar</td>
<td>C++14</td>
</tr>
<tr>
<td>Misra</td>
<td>C++03 (working to C++17)</td>
</tr>
<tr>
<td>High Integrity CPP</td>
<td>C++11</td>
</tr>
<tr>
<td>JSF</td>
<td>C++03</td>
</tr>
<tr>
<td>C++ CG</td>
<td>C++11/14/17/20/latest</td>
</tr>
<tr>
<td>CERT C++</td>
<td>C++14</td>
</tr>
<tr>
<td>Coding Standard</td>
<td>Number of Rules</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Misra C++</td>
<td>229</td>
</tr>
<tr>
<td>High Integrity C++</td>
<td>155</td>
</tr>
<tr>
<td>JSF ++</td>
<td>226</td>
</tr>
<tr>
<td>C++ CG</td>
<td>412</td>
</tr>
<tr>
<td>CERT C++</td>
<td>156</td>
</tr>
</tbody>
</table>

These are the 2 most important guidelines today
C++ CG: Meta + automated checkable rule

- Follow Bjarne’s talk on type+resource Safety on C++ CG
- Aim for bug free code with high performance and elegant coding style
- Meta rules + checkable rules
  - Use GSL, CLion,
- Use a carefully crafted set of programming techniques
  - supported by library facilities
  - enforced by static analysis.
- Available on GitHub

Philosophy

- Express ideas directly in code
- Write in ISO Standard C++
- Express intent
- Ideally, a program should be statically type safe
- Prefer compile-time checking to run-time checking
- What cannot be checked at compile time should be checkable at run time
- Catch run-time errors early
- Don't leak any resources
- Don't waste time or space
- Prefer immutable data to mutable data
- Encapsulate messy constructs, rather than spreading through the code
MISRA and Automated checkable rules with some meta

MISRA – Motor Industry Software Reliability Association

**MISRA C**
- 1998 - Guidelines for the use of the C language in vehicle based software
  - MISRA C:1998 (MISRA C1)
- 2004 - MISRA C:2004 Guidelines for the use of the C language in critical systems
  - MISRA C:2004 (MISRA C2)
- 2013 - MISRA C:2012 Guidelines for the use of the C language in critical systems
  - MISRA C:2012 (MISRA C3)
  - 159 rules of which 138 are statically enforceable

**MISRA C++**
- 2008 - Guidelines for the use of the C++ language in critical systems
  - 228 rules of which 219 are statically enforceable

- Code will always have bug, but they must do no harm
What is still missing?

- So far most only deal with Sequential code
- Very few deal with Parallel code
- Even fewer deal with Concurrent, event driven code
- None deal with Heterogeneous dispatch code

There is always going to be:

- Dirty data, faulty HW, integrity problems
- NEED Freedom from interference, which is much harder in multithread system
- Heterogeneous-> AI/ML safety
Note: this is an early draft WIP. It’s known to be incomplete and incorrect, and it has lots of bad formatting.

Table of Content

0.1 Language Independent Issues

0.1 Language Independent Issues

0.2 General

0.2.1 [1] Think in terms of tasks, rather than threads

0.2.2 [2] Do not use platform specific multi-threading facilities

0.3 Thread

0.3.1 [3] Join std::thread before going out of scope of all locally declared objects passed to thread callable object via pointer or reference. Think of a joining thread as a scoped container

0.3.2 [4] Thread callable object may receive only global and static objects via pointer or reference, if std::thread will be detached. Think of a thread as a global container

0.3.3 [5] Do not use std::thread as a function. Don’t detach a thread

0.3.4 [6] Use high_integrity::thread in place of std::thread

0.3.5 [7] Do not call std::thread::detach() function

0.3.6 [8] Verify resource management assumptions of std::thread with the implementation of standard library of choice

0.1 Language Independent Issues

0.2 General

0.2.1 [1] Think in terms of tasks, rather than threads

0.2.2 [2] Do not use platform specific multi-threading facilities

0.3 Thread

0.3.1 [3] Join std::thread before going out of scope of all locally declared objects passed to thread callable object via pointer or reference. Think of a joining thread as a scoped container

0.3.2 [4] Thread callable object may receive only global and static objects via pointer or reference, if std::thread will be detached. Think of a thread as a global container

0.3.3 [5] Do not use std::thread as a function. Don’t detach a thread

0.3.4 [6] Use high_integrity::thread in place of std::thread

0.3.5 [7] Do not call std::thread::detach() function

0.3.6 [8] Verify resource management assumptions of std::thread with the implementation of standard library of choice

4 Mutex

4.1 [4] Do not call member functions of std::thread

4.1.1 [11] Use std::lock_guard, std::lock::try_lock() or std::lock

4.2 [4] Do not access the members of std::thread

4.2.1 [11] Use and lock guards

4.4.1 [12] Do not destroy objects of the family

4.5.1 [13] Mutexes locked with std::lock or std::lock_guard

4.6.1 [13] Do not call virtual functions and c

4.7.1 [15] Avoid deadlocks by locking in a prioritized order

4.8.1 [16] Objects of std::lock guards, std::lock

4.9.1 [17] Define a mutex together with the data it protects

4.10.1 [18] Do not speculatively lock a non-modifiable

4.11.1 [19] There shall be no code path with


4.13.1 [21] std::recursive_mutex and std::recursive

4.14.1 [21] std::recursive_mutex and std::recursive
Stage 1: extensive deep analysis of 81 rules

- Started in 2019 at a MISRA meeting
  - Why are there no rules for parallelism in MISRA?
- 2019-2021: Phase 1 complete
  - Reviewed 81 rules pulled from
    - C++CG
    - HIC++
    - REphrase H2020 project
    - CERT C++
    - JSF++ (no parallel rules)
    - WG23 (no parallel rules)
    - Added some from our own contributions
- Many joined, average 5-8 per meeting
  - Also consulted outside concurrency and safety experts
- Shared Drive of Phase 1 analysis:
  - https://docs.google.com/document/d/14E0BYqsH_d7fMKvXvaZW0tIC65cYBw0aZp4dlev0Q/edit#heading=h.yt0hxah53p9e
<table>
<thead>
<tr>
<th>Rule</th>
<th>Category</th>
<th>decidable via human review</th>
<th>decidable via tools</th>
<th>status</th>
<th>Destination: Tools vs C++ Core guideline</th>
<th>Reason for keeping and #C++ Core guideline</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2.2 [2]</td>
<td>advisory</td>
<td>easy</td>
<td>partially</td>
<td>consider later</td>
<td>consider later</td>
<td>only partially detactable and use [7] instead</td>
</tr>
<tr>
<td>0.3.1</td>
<td>advisory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3.4 [82]</td>
<td>required</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3.5 [4]</td>
<td>required</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.3 [5]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.8 [7]</td>
<td>required</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.1</td>
<td>advisory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.3</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.4</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.5 [13]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.6</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.9</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.10</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.11</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.12</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.13 [21]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4.14 [22]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.1 [23]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.2 [24]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.3 [25]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.4 [26]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.5 [27]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.6 [28]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.7 [29]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.8 [30]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.9 [31]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.10 [32]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.11 [33]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.12 [34]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.13 [35]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.14 [36]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.15 [37]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.16 [38]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.17 [39]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.18 [40]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.19 [41]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.20 [42]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.21 [43]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.22 [44]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.23 [45]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.24 [46]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.25 [47]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.26 [48]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.27 [49]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.28 [50]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.29 [51]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.30 [52]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.31 [53]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.32 [54]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.33 [55]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.34 [56]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.35 [57]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.36 [58]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.37 [59]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.38 [60]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.39 [61]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.40 [62]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.41 [63]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.42 [64]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.43 [65]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.44 [66]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.45 [67]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.46 [68]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.47 [69]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.48 [70]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.49 [71]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.50 [72]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5.51 [73]</td>
<td>mandatory</td>
<td>complex</td>
<td>yes, on system level</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rule decidability

• Human review
  • Generally simple rules
  • Code snippets
  • Basic syntax matches intention

• Automated tool
  • Static scope: can be convoluted but doable and simple for this generation of tools
  • Dynamic scope: much more complex, hard even for tools of this generation, may be doable with whole program analysis
    • Intention is hidden

• Both Human and Automated tools
  • Generally simple cases
  • Intention is shown in syntax

• Neither are good
  • Very hard cases, dynamic scope, whole program analysis
  • Intention is not clear
    • In these cases we wonder if an [[intention:]] attribute might help
## Where should parallel/concurrency/hetero rules go?

<table>
<thead>
<tr>
<th>Human decidable</th>
<th>Tool decidable</th>
<th>Suitable tools in order of preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>Easy</td>
<td>C++CG, MISRA tools</td>
</tr>
<tr>
<td>Easy</td>
<td>Hard</td>
<td>C++CG, Tools will be meta or undecidable, lots of false positive May be bad rule for tools</td>
</tr>
<tr>
<td>Hard</td>
<td>Easy</td>
<td>MISRA tools, CG Meta</td>
</tr>
<tr>
<td>Hard</td>
<td>Hard</td>
<td>Neither, META directive; Code guidelines Obvious rules, but hard to verify Might not be a good rule anyway Need a new [[intention::]] attribute</td>
</tr>
</tbody>
</table>
Stage 2: collate

- **Category**
  - Mandatory: 8
  - Required: 12
  - Advisory: 12
  - Directive: 5
- **Decidable by humans**
  - Easy: 27
  - Medium: 1
  - Complex: 20
  - Unknown yet: 9
- **Decidable via automated tools**
  - Yes, on a local level: 20
  - Yes, on a system level: 6
  - Maybe, on a system level: 7
  - No: 8
  - Unknown yet: 11
CG, Misra, both or neither

- Accepted: for initial entry 24
  - CG+tools: 12
  - Tools+CG: 5
  - Modifies CG: 4
  - Same as CG: 3
- Deferred for future: 26
- Rejected: 18
- Shared drive of Status from Phase 1:
  - https://docs.google.com/spreadsheets/d/1f-NX2z6axLyv5P0mh4aeNfKO7KLSVSTtZrTwS2YO02M/edit#gid=0
Agenda

1. Current status of C++ safety: MISRA and C++ CG
2. Parallel Safety rules
3. Automotive Safety case
For humans and tools -&gt; C++CG and MISRA

- Rule 13 Mutexes locked with std::lock or std::try_lock shall be wrapped with std::lock_guard, std::unique_lock or std::shared_lock with adopt_lock tag within the same scope

The rule intention is to employing RAII for controlling the state of mutexes in exceptional conditions

Is it easy to detect via review?
- just check std::lock arguments

Is it easy to detect by tool?
- tool can check std::lock arguments

Good for C++CG
Good for MISRA
For humans and tools -> C++CG and MISRA

- Rule 13 Mutexes locked with `std::lock` or `std::try_lock` shall be wrapped with `std::lock_guard`, `std::unique_lock` or `std::shared_lock` with `adopt_lock` tag within the same scope

C++CG has these rules:

**CP.20**: Use RAII, never plain `lock()`/`unlock()`

*Reason* Avoids nasty errors from unreleased locks.

**CP.21**: Use `std::lock()` or `std::scoped_lock` to acquire multiple mutexes

*Reason* To avoid deadlocks on multiple mutexes.

Rule 13 is intentionally friendlier for tools, if compared with CP.21
For humans not tools -> C++CG

• Rule 39 Use std::call_once to ensure a function is called exactly once (rather than the Double-Checked Locking pattern)

The rule intention is to avoid common errors, which might be introduced if common concurrency task is being reproduced with less care

Is it easy to detect via review?
• via careful understanding of the coder intention

Is it easy to detect by tool?
• It is difficult to detect incorrectly written Double-Checked Locking pattern
For humans not tools -> C++CG

• Rule 39 Use std::call_once to ensure a function is called exactly once (rather than the Double-Checked Locking pattern)

C++CG has these rules:

**CP.110: Do not write your own double-checked locking for initialization**

*Reason* Since C++11, static local variables are now initialized in a thread-safe way. When combined with the RAII pattern, static local variables can replace the need for writing your own double-checked locking for initialization. std::call_once can also achieve the same purpose. Use either static local variables of C++11 or std::call_once instead of writing your own double-checked locking for initialization.
For tools not humans -> MISRA

- Rule 19 The order of nested locks/unlock shall form a DAG

The rule intention is to avoid deadlocks via careful tracing of locking and unlocking order.

Is it easy to detect via review?
- Starting from the moderately complex code, it becomes very difficult to trace the order of locks

Is it easy to detect by tool?
- It is theoretically possible, if all the code underneath the specified block is visible

Harder for C++CG

Better for Tools
For humans only -> meta rules

• Rule 8 Verify resource management assumptions of std::thread with the implementation of standard library of choice

Safety implies careful analysis of assumption introduced by dependencies, this one should be reviewed with especial care

Is it easy to detect via review?
• It is not visible in code and should be applied on higher review levels

Too high level for C++CG?

Is it easy to detect by tool?
• It is not visible in code and is not visible for code analysis tools

Directive for tools/MISRA
Agenda

1. Current status of C++ safety: MISRA and C++ CG
2. Parallel Safety rules
3. Automotive Safety case
Why Concurrency guidelines for automotive?

- ISO26262 lists concurrency aspects as one topic to be covered by “Modeling and coding guidelines”
- But should a safety-critical system contain concurrent control-flow at all?

- Typical embedded systems are small and follow static execution patterns
- Even a multi-core automotive chip may have parallel execution but no concurrency if components scheduled on different cores do not interact
The Old World vs. The New

- Traditionally, automotive systems rely on static scheduling
- Each task is given a predetermined time slice in the schedule for execution
- The complete schedule is configured upfront as part of the system design
- Temporal execution of tasks is completely deterministic

⇒ Tasks not scheduled to run in parallel will not overlap. Synchronization between cores is handled by a thin basic software module layer.

⇒ Component interaction across cores is minimized
The Old World vs. The New

- Static scheduling works well only if the number of components is small or the interaction between components is minimal
- With compute-intensive applications like highly automated driving, parallel computation is needed to process data in time

Old: Many small independent applications sharing compute resources of a single CPU. Simple basic software layer at the bottom.

New: One single application requiring all compute resources of a powerful multi-core CPU and possibly a number of auxiliary hardware accelerators. Fully fledged OS at the bottom.

⇒ Lots of concurrency, asynchronous APIs as the default
Asynchronous APIs in Adaptive Autosar

```cpp
ara::com::FindServiceHandle find_service_handle = RadarServiceProxy::StartFindService(
    [](ara::com::ServiceHandleContainer<RadarServiceProxy::HandleType> handles, ara::com::FindServiceHandle) {

    });

    // ...

    RadarServiceProxy::StopFindService(find_service_handle);
```
Asynchronous APIs in Adaptive Autosar

```c
ara::com::FindServiceHandle find_service_handle = RadarServiceProxy::StartFindService(
    [](ara::com::ServiceHandleContainer<RadarServiceProxy::HandleType> handles, ara::com::FindServiceHandle)
    {
        if (handles.empty()) { return; }
        RadarServiceProxy service(handles.front());

        ara::core::Future<uint32_t> fut = service.UpdateRate.Get();

        // ...
    });
// ...
RadarServiceProxy::StopFindService(find_service_handle);
```
Asynchronous APIs in Adaptive Autosar

```c++
ara::com::FindServiceHandle find_service_handle = RadarServiceProxy::StartFindService(
    [](ara::com::ServiceHandleContainer<RadarServiceProxy::HandleType> handles, ara::com::FindServiceHandle)
    { 
        if (handles.empty()) { return; } 
        RadarServiceProxy service(handles.front());

        ara::core::Future<uint32_t> fut = service.UpdateRate.Get();
        auto fut2 = fut.then([](ara::core::Future<uint32_t> f) -> auto {
            uint32_t update_rate = f.get();
            // ...
        });

        // ...
    });

    // ...
}));
// ...
RadarServiceProxy::StopFindService(find_service_handle);
```
Bug Example

```cpp
struct SharedData {

    uint32_t update_rate;

} shared_data;

fut.then([&shared_data](ara::core::Future<uint32_t> f) -> auto {

    shared_data.update_rate = f.get();

});
```
struct SharedData {
    std::mutex mtx;
    uint32_t update_rate;
} shared_data;

fut.then([&shared_data](ara::core::Future<uint32_t> f) -> auto {
    shared_data.mtx.lock();
    shared_data.update_rate = f.get();
});
```cpp
struct SharedData {
    std::mutex mtx;
    uint32_t update_rate;
} shared_data;

fut.then([&shared_data](ara::core::Future<uint32_t> f) -> auto {
    shared_data.mtx.lock();
    shared_data.update_rate = f.get();
    // WARNING: Do not destroy objects of type std::mutex
    // if object is in locked state
});
```
Bug Example

```cpp
struct SharedData {
    std::mutex mtx;
    uint32_t update_rate;
} shared_data;

fut.then([&shared_data](ara::core::Future<uint32_t> f) -> auto {
    shared_data.mtx.lock();
    shared_data.update_rate = f.get();
    shared_data.mtx.unlock();
});
```
struct SharedData {
    std::mutex mtx;
    uint32_t update_rate;
} shared_data;

fut.then([&shared_data](ara::core::Future<uint32_t> f) -> auto {
    shared_data.mtx.lock();
    shared_data.update_rate = f.get();  // get() may throw!
    shared_data.mtx.unlock();
});
Bug Example

```cpp
struct SharedData {
    std::mutex mtx;
    uint32_t update_rate;
}
shared_data;

fut.then([&shared_data](ara::core::Future<uint32_t> f) -> auto {
    shared_data.mtx.lock();  // WARNING: Do not call member functions of std::mutex
    shared_data.update_rate = f.get();  // get() may throw!
    shared_data.mtx.unlock();
});
```
Bug Example

```cpp
struct SharedData {
    std::mutex mtx;
    uint32_t update_rate;
} shared_data;

fut.then([&shared_data](ara::core::Future<uint32_t> f) -> auto {
    std::scoped_lock{ shared_data.mtx };
    shared_data.update_rate = f.get();
});
```
Bug Example

```cpp
struct SharedData {
    std::mutex mtx;
    uint32_t update_rate;
} shared_data;

fut.then([&shared_data](ara::core::Future<uint32_t> f) -> auto {
    std::scoped_lock{ shared_data mtx }; // WARNING: Objects of type std::scoped_lock shall always be named
    shared_data.update_rate = f.get();
});
```
Bug Example

```cpp
struct SharedData {
    std::mutex mtx;
    uint32_t update_rate;
} shared_data;

fut.then([&shared_data](ara::core::Future<uint32_t> f) -> auto {
    std::scoped_lock lk{ shared_data.mtx };
    shared_data.update_rate = f.get();
});
```
Does this mean concurrency is now fine for safety-critical software?

Rules only attempt to catch common pitfalls in using concurrency facilities.

They are only one building block in a larger safety strategy.

Safety implications of use of concurrency must be carefully evaluated in the context of the overall safety strategy.
Final Words
<table>
<thead>
<tr>
<th>Asynchronous Agents</th>
<th>Parallel collections</th>
<th>Mutable shared state</th>
<th>Heterogeneous/Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++11: thread, lambda function, TLS, async</td>
<td>C++11: packaged tasks, promises, futures,</td>
<td>C++11: locks, memory model, mutex, condition variable, atomics, static init/term,</td>
<td>C++11: lambda</td>
</tr>
<tr>
<td>C++ 20: Jthreads +interrupt _token, coroutines</td>
<td>C++ 17: ParallelSTL, control false sharing</td>
<td>C++ 14: shared_lock/shared_timed_mutex, OOTA, atomic_signal_fence,</td>
<td>C++17: progress guarantees, TOE, execution policies</td>
</tr>
<tr>
<td></td>
<td>C++20: Vec execution policy, Algorithm un-sequenced policy</td>
<td>C++ 17: scoped _lock, shared_mutex, ordering of memory models, progress guarantees, TOE, execution policies</td>
<td>C++20: atomic_ref</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Atomic C/C++ compatibility
- Semaphores and waiting
- Fixed gaps in memory model
- Improved atomic flags, Repair memory model
Future safety rules for C++ 20/23 parallelism

• Not inventing, just documenting common wisdom which takes time
• MISRA NEXT is really MISRA 2008 + AUTOSAR
  • Need more manpower
  • Need more experience on the safety of new features
  • Will not cover C++20 and might even miss a few C++17 features
• MISRA parallel will also be in stages
  • C++11 atomics, async .mm
  • C++14 shared lock
  • C++17 parallel algo, futures, (still need more deep dive) unseq,
  • C++20 latches barriers, coroutine, atomic ref,
  • C++23 senders and receivers?
  • C++26 executors networking?, concurrency TS2?
Conclusion and Future plan

• 2021: plan to integrate with MISRA 202X NEXT release
  • 17 rules to MISRA C++ NEXT
  • 17 rules to C++CG
  • 4 CG rules to be modified
  • Reset Deferred to next phase
• 2022-?
  • Work on Deferred rules
  • Add new rules covering
    • Coroutines, parallel algorithm, executors,
  • Aim for next release of MISRA NEXT+ CG NEXT
• Continue with more phases and more releases
C++ Will need to integrate safety with ML

From sequential->concurrency
- 26262
- Adaptive autosar

From concurrency->heterogeneous
- ML /AI trustworthiness, safety
- 21448 SOTIF
- UL4600
- SAE ORAD
CG, Misra, both or neither
- Accepted: for initial entry 24
  - CG+tools: 12
  - Tools+CG: 5
  - Modifies CG: 4
  - Same as CG: 3
- Deferred for future: 26
- Rejected: 18
- Shared drive of Status from Phase 1:
  - https://docs.google.com/spreadsheets/d/1f-NX2z6axLyv5P0mh4aeNfKO7KLXVSTtZrTwS2Y02M/edit#gid=0
Safety Critical API Evolution

minimize API surface area, reduce ambiguity, UB, increase determinism

Industry Need for CPU/GPU Acceleration APIs designed to ease system safety certification

Rendering  Compute  Display