What’s New in SYCLsc™ (SYCL for Safety Critical Systems)

Erik Tomusk, Codeplay Software

Verena Beckham, Codeplay Software
Introduction

• Follow-up to last year’s presentation
  • https://www.youtube.com/watch?v=tD46tGOlUg
Introduction

• Follow-up to last year’s presentation
  • https://www.youtube.com/watch?v=tD46tGOglUg
• Short version:
Introduction

• Follow-up to last year’s presentation
  • https://www.youtube.com/watch?v=tD46tGOglUg
• Short version:
  • Had a Khronos exploratory forum (EF) to investigate demand for and scope of SYCL SC
Introduction

• Follow-up to last year’s presentation
  • https://www.youtube.com/watch?v=tD46tGOglUg
• Short version:
  • Had a Khronos exploratory forum (EF) to investigate demand for and scope of SYCL SC
  • The EF produced a statement of work (SoW)
Introduction

• Follow-up to last year’s presentation
  • https://www.youtube.com/watch?v=tD46tGOglUg
• Short version:
  • Had a Khronos exploratory forum (EF) to investigate demand for and scope of SYCL SC
  • The EF produced a statement of work (SoW)
  • Approved March 2023
Introduction

• Follow-up to last year’s presentation
  • https://www.youtube.com/watch?v=tD46tGOglUg
• Short version:
  • Had a Khronos exploratory forum (EF) to investigate demand for and scope of SYCL SC
  • The EF produced a statement of work (SoW)
  • Approved March 2023
  • EF finished; now there is a working group (WG)
The deployment of autonomous systems is increasingly dependent on open standards and a robust software and hardware ecosystem that places safety as its top priority.

Tom Conway
(Sr. Director, Product Management, Automotive)

Mercedes-Benz Research & Development North America is delighted to join the SYCL SC Working Group to strengthen industry adoption of safety-critical development standards, as ADAS/AD systems harness heterogeneous computing hardware.

Sundararajan Ramalingam
(VP Autonomous Driving)

SYCL Safety Critical extends open accelerator programming to applications where safety critical standards apply.

Joe Curly
(VP Software Products)

Outline
Outline

1. Background
2. Summary of SYCL SC EF activities
3. Technical lessons learned
   • SYCL SC Scope
   • SC dynamic memory
4. Conclusions
Outline

1. Background
2. Summary of SYCL SC EF activities
3. Technical lessons learned
   • SYCL SC Scope
   • SC dynamic memory
4. Conclusions

• Caveats:
  • Speaking as a Codeplay employee
  • Not representing Khronos
  • Not representing SYCL SC EF or SYCL SC WG
  • SYCL SC EF was under NDA
Outline

1. Background
2. Summary of SYCL SC EF activities
3. Technical lessons learned
   • SYCL SC Scope
   • SC dynamic memory
4. Conclusions
Background: “Safety Critical”
Background: “Safety Critical”

• For software, a safety-critical domain is one where a software malfunction could cause serious damage
  • To people
  • To property
  • To the environment
Background: “Safety Critical”

- For software, a safety-critical domain is one where a software malfunction could cause serious damage
  - To people
  - To property
  - To the environment
- Domains include
  - Automotive
  - Aerospace
  - Medical
  - Nuclear
  - Rail
  - …
Background: Why SYCL?
Background: Why SYCL?

- Accelerator
- Low-Level API
Background: Why SYCL?

- **Accelerator**: Much faster than CPU for some tasks
- **Low-Level API**
Background: Why SYCL?

Accelerator

Much faster than CPU for some tasks

Low-Level API
Background: Why SYCL?

- Accelerator
  - Much faster than CPU for some tasks
- Low-Level API
- Proprietary
- Level Zero
- OpenCL
- Vulkan

Accelerator types:
- FPGA
- ASIC
- GPU
Background: Why SYCL?

- Accelerator: Much faster than CPU for some tasks
- Proprietary
- Level Zero
- OpenCL
- Verbose
- Complicated
- Benefit from hardware expertise
Background: Why SYCL?

- Accelerator
  - FPGA
  - ASIC
  - GPU
  - Much faster than CPU for some tasks

- Low-Level API
  - Proprietary
  - Level Zero
  - OpenCL
  - Vulkan

- Framework
- Application
  - Complicated
  - Benefit from hardware expertise
  - Verbose

Beneath: Proprietary

Complicated
Background: Why SYCL?

Accelerator
- FPGA
- ASIC
- GPU

Low-Level API
- Proprietary
- Level Zero
- OpenCL
- Vulkan

Framework
- Applications need acceleration
- Developers are domain experts, not hardware experts

Application

Much faster than CPU for some tasks
- Verbose
- Complicated
- Benefit from hardware expertise
Developers are domain experts, not hardware experts

- PyTorch (ML)
- NAMD (molecular dynamics)
- Cycles (Blender)

Applications need acceleration

Developers are domain experts, not hardware experts

- Much faster than CPU for some tasks
- Verbose
- Complicated
- Benefit from hardware expertise

Proprietary

Level Zero
Background: Why SYCL?

Accelerator → Low-Level API → Framework → Application

- FPGA
- ASIC
- GPU

- Level Zero
- Proprietary
- OpenCL
- Vulkan

- Much faster than CPU for some tasks
- Verbose
- Complicated
- Benefit from hardware expertise
- Abstraction gap
- Requires detailed knowledge of framework and low-level API
- Requires re-implementing framework on every low-level API

- PyTorch (ML)
- NAMD (molecular dynamics)
- Cycles (Blender)

- Applications need acceleration
- Developers are domain experts, not hardware experts

Developers are domain experts, not hardware experts
Background: Why SYCL?

Accelerator
- Much faster than CPU for some tasks

Low-Level API
- Verbose
- Complicated
- Benefit from hardware expertise

Framework
- PyTorch (ML)
- NAMD (molecular dynamics)
- Cycles (Blender)

Application
- Developers are domain experts, not hardware experts
- Applications need acceleration

Developers are domain experts, not hardware experts
Background: Why SYCL?

Accelerator
- Much faster than CPU for some tasks

Low-Level API
- Verbose
- Complicated
- Benefit from hardware expertise

SYCL
- Familiar C++
- Abstract away low-level details
- Developer quality of life features

Framework
- PyTorch (ML)
- NAMD (molecular dynamics)
- Cycles (Blender)

Application
- Applications need acceleration
- Developers are domain experts, not hardware experts

Developers are domain experts, not hardware experts

Background: Why SYCL?

Accelerator
- Much faster than CPU for some tasks

Low-Level API
- Verbose
- Complicated
- Benefit from hardware expertise

SYCL
- Familiar C++
- Abstract away low-level details
- Developer quality of life features

Framework
- PyTorch (ML)
- NAMD (molecular dynamics)
- Cycles (Blender)

Application
- Applications need acceleration
- Developers are domain experts, not hardware experts

Developers are domain experts, not hardware experts
Background: Why SYCL SC?

- FPGA
- ASIC
- GPU

Accelerator <-> Low-Level API
Framework <-> Application
Background: Why SYCL SC?

- FPGA
- ASIC
- GPU

Proprietary

Low-Level API

Accelerator

Framework

Application
Background: Why SYCL SC?

SC applications need acceleration
SC applications need greater portability
Background: Why SYCL SC?

Accelerator

Low-Level API

Frameworks not viable

Abstraction gap

Certification overhead

SC applications need acceleration

SC applications need greater portability

FPGA
ASIC
GPU

Proprietary

OpenGL/SC

Vulkan/SC

Application

Framework

STOP
Background: Why SYCL SC?

SC applications need greater portability
SC applications need acceleration

Accelerator
Low-Level API
Proprietary

FPGA
ASIC
GPU

SYCL

Framework
Application
Previously infeasible applications become possible.

SC applications need greater portability.

Well-designed API can reduce certification effort.

Previously infeasible applications become possible.

Background: Why SYCL SC?
Background: Why not just SYCL?
Background: Why not just SYCL?
Background: Why not just SYCL?

ACCELERATOR

Low-Level API

SYCL

SC Framework

Application

SC Processes

ISO 26262 (Automotive)
ISO 21448 (SOTIF)
DO-178C (Avionics)
IEC 62304 (Medical)
EN 50128 (Rail)
Background: Why not just SYCL?

SYCL SC needs to be compatible with SC processes

SC Processes

ISO 26262 (Automotive)
ISO 21448 (SOTIF)
IEC 62304 (Medical)
DO-178C (Avionics)
EN 50128 (Rail)

Accelerator ↔ Low-Level API ↔ SYCL SC ↔ Framework ↔ Application
Background: Why not just SYCL?

SYCL SC needs to be compatible with SC processes.
Background: Why not just SYCL?

SYCL SC needs to be compatible with SC processes.
Background: Why not just SYCL?

SYCL SC needs to be compatible with SC processes

SYCL SC needs to be compatible with SC code guidelines

SC Processes

SC Code Guidelines

Accelerator

Low-Level API

SYCL SC

Framework

Application

ISO 26262 (Automotive)

DO-178C (Avionics)

EN 50128 (Rail)

ISO 21448 (SOTIF)

IEC 62304 (Medical)

AUTOSAR C++

MISRA

JSF++
Background: Why not just SYCL?
Background: Why not just SYCL?

- SYCL SC philosophy
Background: Why not just SYCL?

• SYCL SC philosophy
  1. Enable predictable applications
     • Execution time, resource usage, results
Background: Why not just SYCL?

- SYCL SC philosophy
  1. Enable predictable applications
     - Execution time, resource usage, results
  2. Streamlined functionality to simplify runtime
     - Better compatibility with safety standards
Background: Why not just SYCL?

• SYCL SC philosophy
  1. Enable predictable applications
     • Execution time, resource usage, results
  2. Streamlined functionality to simplify runtime
     • Better compatibility with safety standards
  3. Specify comprehensive error handling, identify and remove ambiguities, and clarify undefined behavior
     • Better compatibility with safety-critical use cases
Background: Why not just SYCL?

• SYCL SC philosophy
  1. Enable predictable applications
     • Execution time, resource usage, results
  2. Streamlined functionality to simplify runtime
     • Better compatibility with safety standards
  3. Specify comprehensive error handling, identify and remove ambiguities, and clarify undefined behavior
     • Better compatibility with safety-critical use cases
• First-order priorities for SYCL SC, but lower priorities for SYCL
Background: What SYCL SC is not?
Background: What SYCL SC is not?

• SYCL SC intends to be *compatible* with SC processes and guidelines
Background: What SYCL SC is not?

- SYCL SC intends to be *compatible* with SC processes and guidelines
- The SYCL SC WG will *not* ...
Background: What SYCL SC is not?

- SYCL SC intends to be *compatible* with SC processes and guidelines
- The SYCL SC WG will *not* …
  - Tell you how to implement a “safe” application
  - Using SYCL SC does not guarantee a safe application
Background: What SYCL SC is not?

- SYCL SC intends to be *compatible* with SC processes and guidelines
- The SYCL SC WG will *not* ...
  - Tell you how to implement a “safe” application
    - Using SYCL SC does not guarantee a safe application
  - Tell you how to implement a “safe” SYCL runtime
    - Implementing the SYCL SC specification does not guarantee a safe runtime
Background: What SYCL SC is not?

- SYCL SC intends to be compatible with SC processes and guidelines
- The SYCL SC WG will not …
  - Tell you how to implement a “safe” application
    - Using SYCL SC does not guarantee a safe application
  - Tell you how to implement a “safe” SYCL runtime
    - Implementing the SYCL SC specification does not guarantee a safe runtime
  - Tell you how to apply any industry process or standard
Background: What SYCL SC is not?

- SYCL SC intends to be *compatible* with SC processes and guidelines
- The SYCL SC WG will *not* …
  - Tell you how to implement a “safe” application
    - Using SYCL SC does not guarantee a safe application
  - Tell you how to implement a “safe” SYCL runtime
    - Implementing the SYCL SC specification does not guarantee a safe runtime
  - Tell you how to apply any industry process or standard
  - Produce any certified code (tools or runtime)
Background: What SYCL SC is not?

- SYCL SC intends to be compatible with SC processes and guidelines
- The SYCL SC WG will not ...
  - Tell you how to implement a “safe” application
    - Using SYCL SC does not guarantee a safe application
  - Tell you how to implement a “safe” SYCL runtime
    - Implementing the SYCL SC specification does not guarantee a safe runtime
  - Tell you how to apply any industry process or standard
  - Produce any certified code (tools or runtime)
    - Code snippets in SYCL SC specification will need to be certifiable
Background: What SYCL SC is not?

- SYCL SC intends to be *compatible* with SC processes and guidelines
- The SYCL SC WG will *not* ...
  - Tell you how to implement a “safe” application
    - Using SYCL SC does not guarantee a safe application
  - Tell you how to implement a “safe” SYCL runtime
    - Implementing the SYCL SC specification does not guarantee a safe runtime
  - Tell you how to apply any industry process or standard
  - Produce any certified code (tools or runtime)
    - Code snippets in SYCL SC specification will need to be *certifiable*
  - SYCL SC will be *compatible* with you doing the above, but cannot do it for you
Outline

1. Background
2. Summary of SYCL SC EF activities
3. Technical lessons learned
   • SYCL SC Scope
   • SC dynamic memory
4. Conclusions
SYCL SC EF: Process

SYCL SC EF

SYCL SC WG

Ideas

SYCL SC 1.0 Specification
**SYCL SC EF: Process**

04/22: SYCL SC EF starts; led by CoreAVI & Codeplay

SYCL SC 1.0 Specification
SYCL SC EF: Process

• Open invitation to all interested parties to present and discuss

04/22: SYCL SC EF starts; led by CoreAVI & Codeplay

SYCL SC 1.0 Specification

Ideas
SYCL SC EF: Process

- Open invitation to all interested parties to present and discuss
- Collate unfiltered wishlist of potential SYCL SC features
SYCL SC EF: Process

- Open invitation to all interested parties to present and discuss
- Collate unfiltered wishlist of potential SYCL SC features
  - **Goal:** Determine if there is a need for a standard like SYCL SC

04/22: SYCL SC EF starts; led by CoreAVI & Codeplay
04/22: SYCL SC EF starts; led by CoreAVI & Codeplay

- Open invitation to all interested parties to present and discuss
- Collate unfiltered wishlist of potential SYCL SC features
  - **Goal:** Determine if there is a need for a standard like SYCL SC
  - **Goal:** Understand priorities
SYCL SC EF: Process

• Open invitation to all interested parties to present and discuss
• Collate unfiltered wishlist of potential SYCL SC features
  • **Goal:** Determine if there is a need for a standard like SYCL SC
  • **Goal:** Understand priorities
  • **Goal:** Identify other requirements

04/22: SYCL SC EF starts; led by CoreAVI & Codeplay
**SYCL SC EF: Process**

**04/22:** SYCL SC EF starts; led by CoreAVI & Codeplay

Collate feature wishlist

**SYCL SC 1.0 Specification**
04/22: SYCL SC EF starts; led by CoreAVI & Codeplay

Collate feature wishlist

Narrow down wishlist to coherent set of requirements

SYCL SC 1.0 Specification

Collate feature wishlist
The SYCL SC EF: Process

04/22: SYCL SC EF starts; led by CoreAVI & Codeplay

- Collate feature wishlist
- Narrow down wishlist to coherent set of requirements
- Justification for a separate standard

SYCL SC 1.0 Specification
**SYCL SC EF: Process**

**04/22**: SYCL SC EF starts; led by CoreAVI & Codeplay

- **Collate feature wishlist**

- **Statement of work**

- **Narrow down wishlist to coherent set of requirements**

- **Justification for a separate standard**

**SYCL SC 1.0 Specification**
**SYCL SC EF: Process**

**04/22:** SYCL SC EF starts; led by CoreAVI & Codeplay

- Collate feature wishlist

**Statement of work**

- Narrow down wishlist to coherent set of requirements
- Justification for a separate standard

**03/23:** Khronos board approves creation of SYCL SC working group

**SYCL SC 1.0 Specification**
SYCL SC EF: Process

04/22: SYCL SC EF starts; led by CoreAVI & Codeplay
   Collate feature wishlist

Statement of work
   Narrow down wishlist to coherent set of requirements
   Justification for a separate standard

03/23: Khronos board approves creation of SYCL SC working group

Today

SYCL SC EF

SYCL SC WG

SyCL SC 1.0 Specification

Ideas
Outline

1. Background
2. Summary of SYCL SC EF activities
3. Technical lessons learned
   • SYCL SC Scope
   • SC dynamic memory
4. Conclusions
Technical Takeaway: SYCL SC Scope
Technical Takeaway: SYCL SC Scope

• Plenty of discussion around the “usual suspects,” like exceptions
Technical Takeaway: SYCL SC Scope

• Plenty of discussion around the “usual suspects,” like exceptions
• Wishlist also contained some more surprising items
Technical Takeaway: SYCL SC Scope

- Plenty of discussion around the “usual suspects,” like exceptions
- Wishlist also contained some more surprising items

Wishlist:
- ...
- Access to accelerators
- ...
- Kernel execution on CPU
- ...
- Hardware-agnostic
- ...
- Other languages? C? Rust?
- ...

SYCL SC EF
Technical Takeaway: SYCL SC Scope

- Plenty of discussion around the “usual suspects,” like exceptions
- Wishlist also contained some more surprising items
  - Led to extended discussions

Wishlist:
- ...
- Access to accelerators
- ...
- Kernel execution on CPU
- ...
- Hardware-agnostic
- ...
- Other languages? C? Rust?
- ...

© 2023 Codeplay Software Ltd.
Technical Takeaway: SYCL SC Scope

• Plenty of discussion around the “usual suspects,” like exceptions
• Wishlist also contained some more surprising items
  • Led to extended discussions
• EF: *Divergence between SYCL and SYCL SC should be minimized*

Wishlist:
- ...
- Access to accelerators
- ...
- Kernel execution on CPU
- ...
- Hardware-agnostic
- ...
- Other languages? C? Rust?
- ...
Technical Takeaway: SYCL SC Scope

- Plenty of discussion around the “usual suspects,” like exceptions
- Wishlist also contained some more surprising items
  - Led to extended discussions
- EF: *Divergence between SYCL and SYCL SC should be minimized*
  - Features that are essential to SYCL do not need to be called out in the SoW

**Wishlist:**
- ...
- Access to accelerators
- ...
- Kernel execution on CPU
- ...
- Hardware-agnostic
- ...
- Other languages? C? Rust?
- ...
Technical Takeaway: SYCL SC Scope

- Plenty of discussion around the “usual suspects,” like exceptions
- Wishlist also contained some more surprising items
  - Led to extended discussions
- EF: Divergence between SYCL and SYCL SC should be minimized
  - Features that are essential to SYCL do not need to be called out in the SoW
  - Features that would diverge SYCL SC from SYCL are excluded from the SoW

Wishlist:
- ...
- Access to accelerators
- ...
- Kernel execution on CPU
- ...
- Hardware-agnostic
- ...
- Other languages? C? Rust?
- ...

© 2023 Codeplay Software Ltd.
Technical Takeaway: SYCL SC Scope

- Plenty of discussion around the “usual suspects,” like exceptions
- Wishlist also contained some more surprising items
  - Led to extended discussions
- EF: *Divergence between SYCL and SYCL SC should be minimized*
  - Features that are essential to SYCL do not need to be called out in the SoW
  - Features that would diverge SYCL SC from SYCL are excluded from the SoW
- These wishlist items aren’t in the SYCL SC requirements

Wishlist:
- ...
- Access to accelerators
- ...
- Kernel execution on CPU
- ...
- Hardware-agnostic
- ...
- Other languages? C? Rust?
- ...
Technical Takeaway: Dynamic Memory
Technical Takeaway: Dynamic Memory

SYCL SC EF  SYCL SC WG

SYCL SC 1.0 Specification
Technical Takeaway: Dynamic Memory

Wishlist:
- ...
- (No) dynamic memory
- ...

SYCL SC EF

SYCL SC WG

SYCL SC 1.0 Specification
Technical Takeaway: Dynamic Memory
Technical Takeaway: Dynamic Memory

• Need to cut through FUD
Technical Takeaway: Dynamic Memory

• Need to cut through FUD
• Q: What, precisely, is bad about dynamic memory?
Technical Takeaway: Dynamic Memory

• Need to cut through FUD
• Q: What, precisely, is bad about dynamic memory?
• A: Non-determinism
Technical Takeaway: Dynamic Memory

• Need to cut through FUD
• **Q:** What, precisely, is bad about dynamic memory?
• **A:** Non-determinism
  • `malloc()` crosses from user space into the OS
Technical Takeaway: Dynamic Memory

• Need to cut through FUD

• **Q:** What, precisely, is bad about dynamic memory?

• **A:** Non-determinism
  
  • `malloc()` crosses from user space into the OS
  
  • `malloc()` takes an unpredictable amount of time to find memory
**Technical Takeaway: Dynamic Memory**

- Need to cut through FUD
- **Q:** What, precisely, is bad about dynamic memory?
- **A:** Non-determinism
  - `malloc()` crosses from user space into the OS
  - `malloc()` takes an unpredictable amount of time to find memory
  - `malloc()` is allowed to return `NULL`
Technical Takeaway: Dynamic Memory

• Need to cut through FUD
• Q: What, precisely, is bad about dynamic memory?
• A: Non-determinism
  • `malloc()` crosses from user space into the OS
  • `malloc()` takes an unpredictable amount of time to find memory
  • `malloc()` is allowed to return NULL
• Safety-critical applications have strict deadlines
Technical Takeaway: Dynamic Memory

• Need to cut through FUD
• Q: What, precisely, is bad about dynamic memory?
• A: Non-determinism
  • `malloc()` crosses from user space into the OS
  • `malloc()` takes an unpredictable amount of time to find memory
  • `malloc()` is allowed to return NULL
• Safety-critical applications have strict deadlines
  • If deadlines can be missed in unpredictable ways, then safety certification becomes difficult
Technical Takeaway: Dynamic Memory

- Need to cut through FUD
- Q: What, precisely, is bad about dynamic memory?
- A: Non-determinism
  - `malloc()` crosses from user space into the OS
  - `malloc()` takes an unpredictable amount of time to find memory
  - `malloc()` is allowed to return NULL
- Safety-critical applications have strict deadlines
  - If deadlines can be missed in unpredictable ways, then safety certification becomes difficult
- Problem isn’t dynamic memory; problem is non-determinism
Technical Takeaway: Dynamic Memory
Technical Takeaway: Dynamic Memory

• Dynamic memory can be made deterministic:
Technical Takeaway: Dynamic Memory

• Dynamic memory can be made deterministic:

• **Solution:** Ensure all calls to `malloc()` happen before application enters time-critical phase
Technical Takeaway: Dynamic Memory

- Dynamic memory can be made deterministic:
- **Solution:** Ensure all calls to `malloc()` happen before application enters time-critical phase
- **Solution:** Don’t call `free()`
Technical Takeaway: Dynamic Memory

• Dynamic memory can be made deterministic:
• **Solution:** Ensure all calls to `malloc()` happen before application enters time-critical phase
• **Solution:** Don’t call `free()`
  • Avoids most complexity with `malloc()`
Technical Takeaway: Dynamic Memory

- Dynamic memory can be made deterministic:
- **Solution:** Ensure all calls to `malloc()` happen before application enters time-critical phase
- **Solution:** Don’t call `free()`
  - Avoids most complexity with `malloc()`
  - Only possible for some applications
Technical Takeaway: Dynamic Memory

- Dynamic memory can be made deterministic:
  - **Solution:** Ensure all calls to `malloc()` happen before application enters time-critical phase
  - **Solution:** Don’t call `free()`
    - Avoids most complexity with `malloc()`
    - Only possible for some applications
  - **Solution:** Use object pools
Technical Takeaway: Dynamic Memory

- Dynamic memory can be made deterministic:
- **Solution:** Ensure all calls to `malloc()` happen before application enters time-critical phase
- **Solution:** Don’t call `free()`
  - Avoids most complexity with `malloc()`
  - Only possible for some applications
- **Solution:** Use object pools
  - Hard upper bound for number of objects
Technical Takeaway: Dynamic Memory

• Dynamic memory can be made deterministic:
  • **Solution**: Ensure all calls to `malloc()` happen before application enters time-critical phase
  • **Solution**: Don’t call `free()`
    • Avoids most complexity with `malloc()`
    • Only possible for some applications
  • **Solution**: Use object pools
    • Hard upper bound for number of objects
    • Vulkan SC solution
Technical Takeaway: Dynamic Memory

- Dynamic memory can be made deterministic:
  - **Solution:** Ensure all calls to `malloc()` happen before application enters time-critical phase
  - **Solution:** Don’t call `free()`
    - Avoids most complexity with `malloc()`
    - Only possible for some applications
  - **Solution:** Use object pools
    - Hard upper bound for number of objects
    - Vulkan SC solution
  - **Solution:** Use local (stack) buffers
Technical Takeaway: Dynamic Memory

- Dynamic memory can be made deterministic:
  - **Solution:** Ensure all calls to `malloc()` happen before application enters time-critical phase
  - **Solution:** Don't call `free()`
    - Avoids most complexity with `malloc()`
    - Only possible for some applications
  - **Solution:** Use object pools
    - Hard upper bound for number of objects
    - Vulkan SC solution
  - **Solution:** Use local (stack) buffers
  - **Solution:** Others?
Technical Takeaway: Dynamic Memory

• Dynamic memory can be made deterministic:

  • Solution: Ensure all calls to `malloc()` happen before application enters time-critical phase
  • Solution: Don’t call `free()`
    • Avoids most complexity with `malloc()`
    • Only possible for some applications
  • Solution: Use object pools
    • Hard upper bound for number of objects
    • Vulkan SC solution
  • Solution: Use local (stack) buffers
  • Solution: Others?

• Conclusion: Deterministic memory management is possible
Technical Takeaway: Dynamic Memory

Wishlist:
- ...
- (No) dynamic memory
- ...

Requirements:
- ...
- Deterministic memory management
- ...

SYCL SC EF

Statement of work

SYCL SC 1.0 Specification

SYCL SC WG
Technical Takeaway: Dynamic Memory
Technical Takeaway: Dynamic Memory

• Many issues like dynamic memory
Technical Takeaway: Dynamic Memory

• Many issues like dynamic memory
• SYCL SC WG will need to…
Technical Takeaway: Dynamic Memory

• Many issues like dynamic memory
• SYCL SC WG will need to…
  • Cut through uncertainty to understand underlying safety concern
Technical Takeaway: Dynamic Memory

- Many issues like dynamic memory
- SYCL SC WG will need to...
  - Cut through uncertainty to understand underlying safety concern
  - Identify possible solutions
Technical Takeaway: Dynamic Memory

• Many issues like dynamic memory
• SYCL SC WG will need to…
  • Cut through uncertainty to understand underlying safety concern
  • Identify possible solutions
  • Make changes to SYCL spec so that SYCL SC is compatible with solutions
Technical Takeaway: Dynamic Memory

• Many issues like dynamic memory
• SYCL SC WG will need to…
  • Cut through uncertainty to understand underlying safety concern
  • Identify possible solutions
  • Make changes to SYCL spec so that SYCL SC is compatible with solutions
• As much as possible, SYCL SC will not mandate a specific solution
Outline

1. Background
2. Summary of SYCL SC EF activities
3. Technical lessons learned
   • SYCL SC Scope
   • SC dynamic memory
4. Conclusions
Call to Action
Call to Action

- Summary
  - What is SC
  - Need for SYCL SC
  - SYCL SC EF overview
  - Technical Takeaways
Call to Action

• Summary
  • What is SC
  • Need for SYCL SC
  • SYCL SC EF overview
  • Technical Takeaways
• Join the SYCL SC working group
Call to Action

• Summary
  • What is SC
  • Need for SYCL SC
  • SYCL SC EF overview
  • Technical Takeaways
• Join the SYCL SC working group
  • Join Khronos
Call to Action

• Summary
  • What is SC
  • Need for SYCL SC
  • SYCL SC EF overview
  • Technical Takeaways
• Join the SYCL SC working group
  • Join Khronos
• Planning an advisory panel for Khronos non-members
Call to Action

• Summary
  • What is SC
  • Need for SYCL SC
  • SYCL SC EF overview
  • Technical Takeaways
• Join the SYCL SC working group
  • Join Khronos
• Planning an advisory panel for Khronos non-members
  • Get in touch if interested
Call to Action

• Summary
  • What is SC
  • Need for SYCL SC
  • SYCL SC EF overview
  • Technical Takeaways
• Join the SYCL SC working group
  • Join Khronos
• Planning an advisory panel for Khronos non-members
  • Get in touch if interested
• Just talk to us
What’s New in SYCL for Safety Critical Systems

Erik Tomusk, Codeplay Software  
erik@codeplay.com

Verena Beckham, Codeplay Software  
verena@codeplay.com