14th International Workshop on OpenCL and SYCL

IWOCL 2026

Safety-Oriented GPU Programming via Language Restriction

Marcos Rodriguez, UPC & Ikerlan Reaseach Center

Jon Pedernales, UPC & Ikerlan Research Center

Leonidas Kosmidis, BSC & UPC

Alejandro J. Calderon, Ikerlan Reseach Center

Irune Yarza, Ikerlan Research Center

May 6-8, 2026 | Heilbronn, Germany | iwocl.org

Content

1. Introduction
2. Proposed approach
3. Use case description
4. Experimental setup
5. Results
6. Conclusions

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

COPYRIGHT, CONFIDENTIAL AND PROPRIETARY. ALL RIGHTS RESERVED - PROPERTY OF IKERLAN, S. COOP. This information carrier and the information it contains are the property of Ikerlan, S. Coop. Any reproduction, disclosure or use of either is prohibited without the prior written consent of Ikerlan, S. Coop. Ikerlan, S. Coop. reserves worldwide all rights also in the case of industrial property rights being granted. The same provisions apply to any oral communications related thereto accordingly

1. Introduction

Introduction

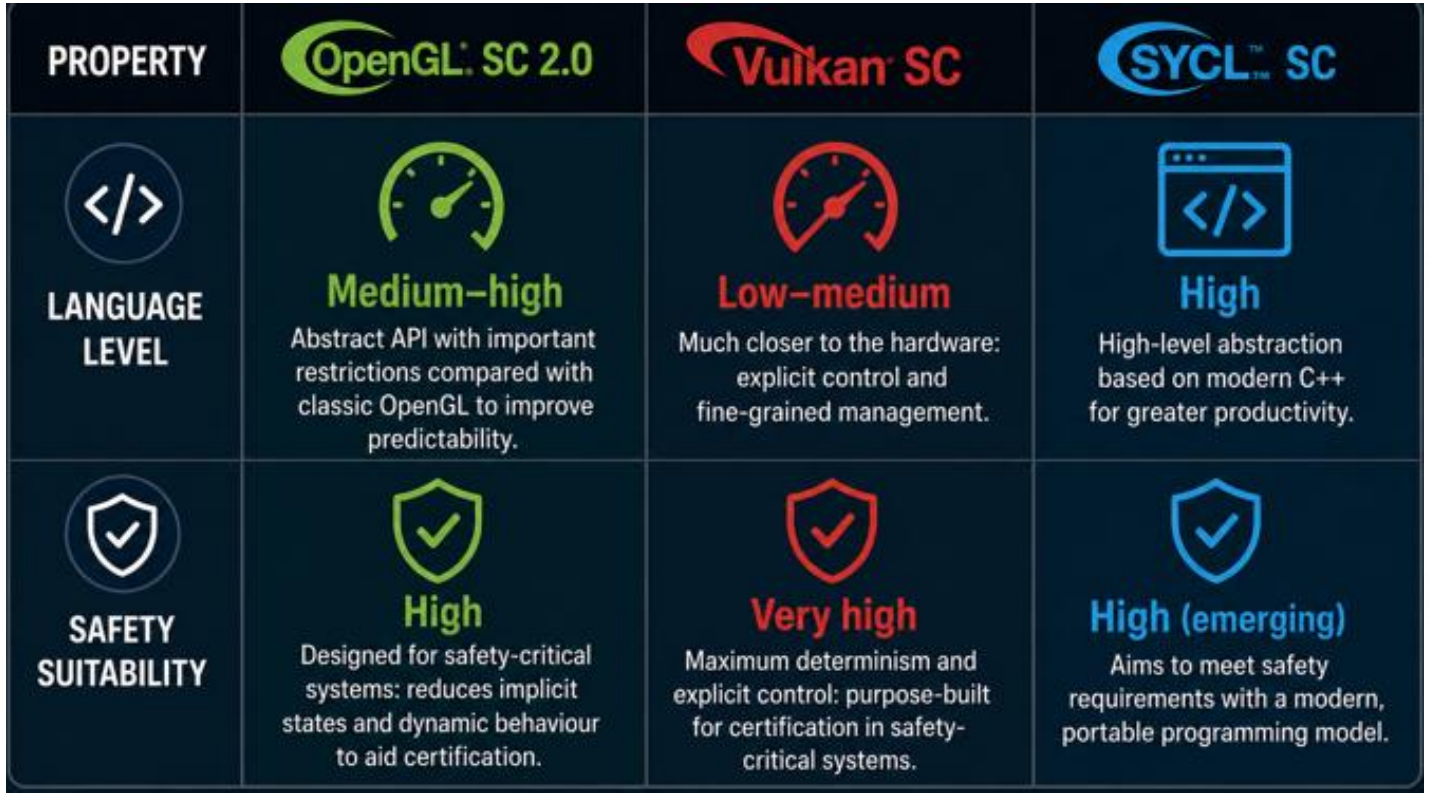
- Emerging safety-critical applications increasingly rely on GPU-based computation to meet performance requirements.

- However, existing GPU software development workflows are not fully aligned with functional safety standards such as IEC 61508 and ISO 26262

Introduction

- Reasons for safety-critical misalignment:
  - GPUs are known about their closed source and **non-deterministic nature**, which complicates their implementation on safety compliant systems.
  - Commonly used language abstractions:
    - Performance - driven
    - Introduce uncertainties and unsafe coding

State of the art in safety-compliant GPU languages



PROPERTY	OpenGL SC 2.0	Vulkan SC	SYCL SC
LANGUAGE LEVEL	**Medium–high** Abstract API with important restrictions compared with classic OpenGL to improve predictability.	**Low–medium** Much closer to the hardware: explicit control and fine-grained management.	**High** High-level abstraction based on modern C++ for greater productivity.
SAFETY SUITABILITY	**High** Designed for safety-critical systems: reduces implicit states and dynamic behaviour to aid certification.	**Very high** Maximum determinism and explicit control: purpose-built for certification in safety-critical systems.	**High (emerging)** Aims to meet safety requirements with a modern, portable programming model.

Starting point

NVIDIA CUDA

[1] SYCLomatic

SYCL™

PROPERTY	CUDA	SYCL
LANGUAGE LEVEL	**Low–medium** Extensions to C/C++ with GPU-specific concepts and CUDA APIs.	**High** High-level, modern C++ standard for heterogeneous programming.
SUPPORTED BY	**NVIDIA** Proprietary technology supported by NVIDIA GPUs and tools.	**Khronos Group** Open standard supported by multiple vendors and wide ecosystem.
HW ABSTRACTION	**Low** Low-level access to NVIDIA hardware; vendor-specific abstractions.	**High** High-level abstraction for portability across CPUs, GPUs, FPGAs and accelerators.
SAFETY SUITABILITY	**Medium** Used in safety-critical systems but not designed specifically for functional safety.	**High (emerging)** Designed with safety in mind; structured model and growing ecosystem for safety-critical use.

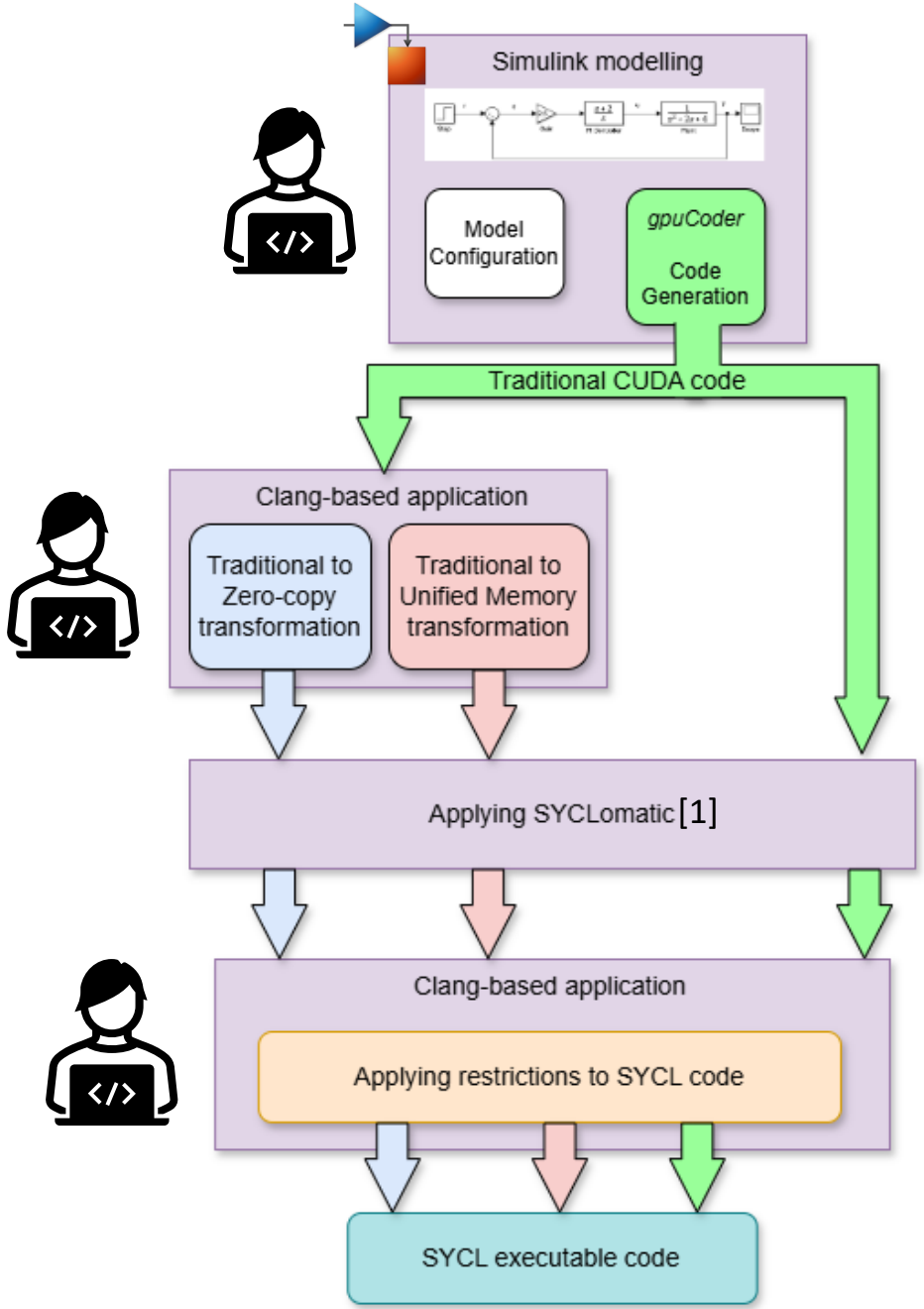
ikerlan MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE · UPC · BSC

[1] Robert Mueller-Albrecht. 2024. SYCLomatic: SYCL Adoption for Everyone - Moving from CUDA to SYCL Gets Progressively Easier: Advanced Migration Considerations. In Proceedings of the 12th International Workshop on OpenCL and SYCL (Chicago, IL, USA) (IWOCL '24)

Contribution

- Workflow from a Simulink model to high-level and safety-compliant language

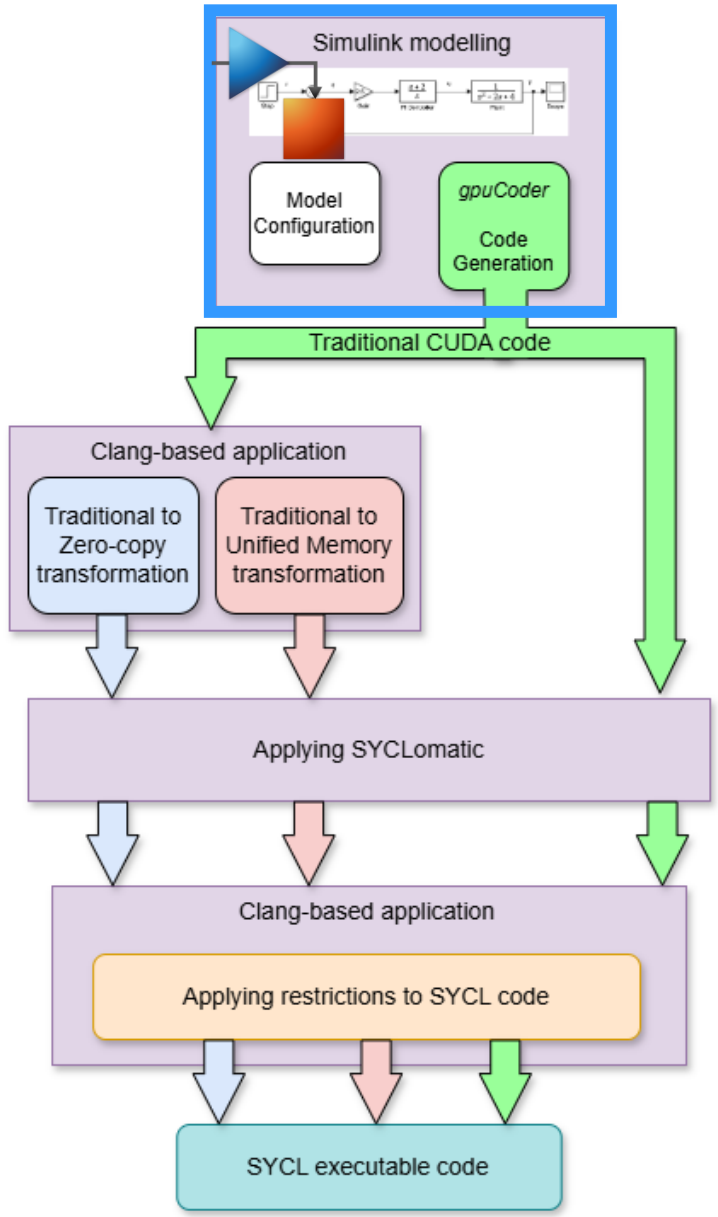
- Added:
  - Model configuration
  - Memory transformation
  - Language restrictions to generated SYCL



2. Proposed approach: Description.

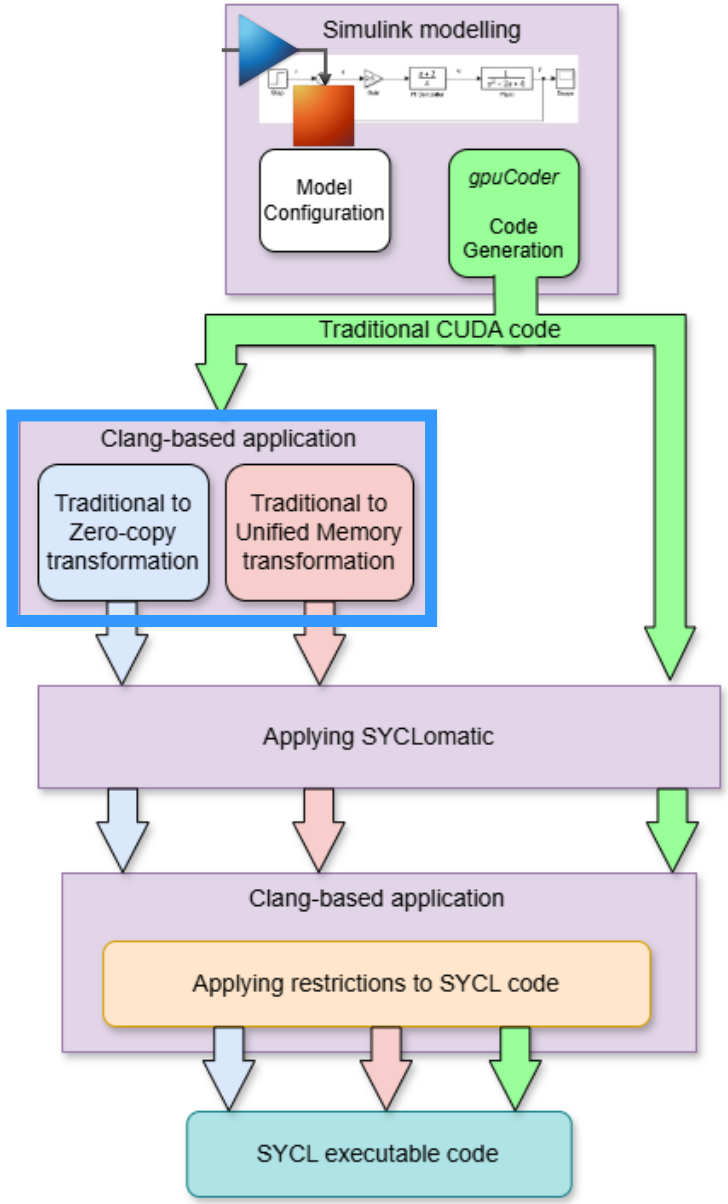
Workflow properties

- Analysability
- Use of high-level language
- Easy to learn
- Verifiability & Testability
- Traceability
- Bounded memory behavior
- Portability / HW abstraction
- Avoidance of unsafe language constructs



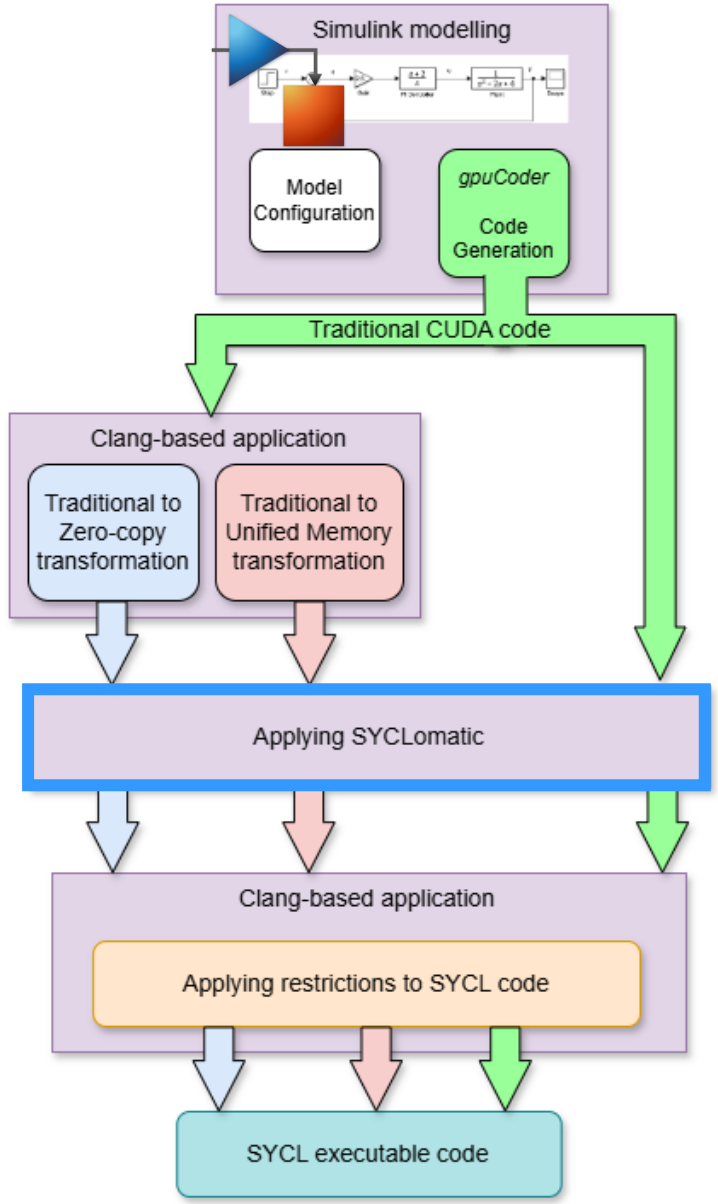
Workflow properties

- Analysability
- Use of high-level language
- Easy to learn
- Verifiability & Testability
- Traceability
- Bounded memory behavior
- Portability / HW abstraction
- Avoidance of unsafe language constructs



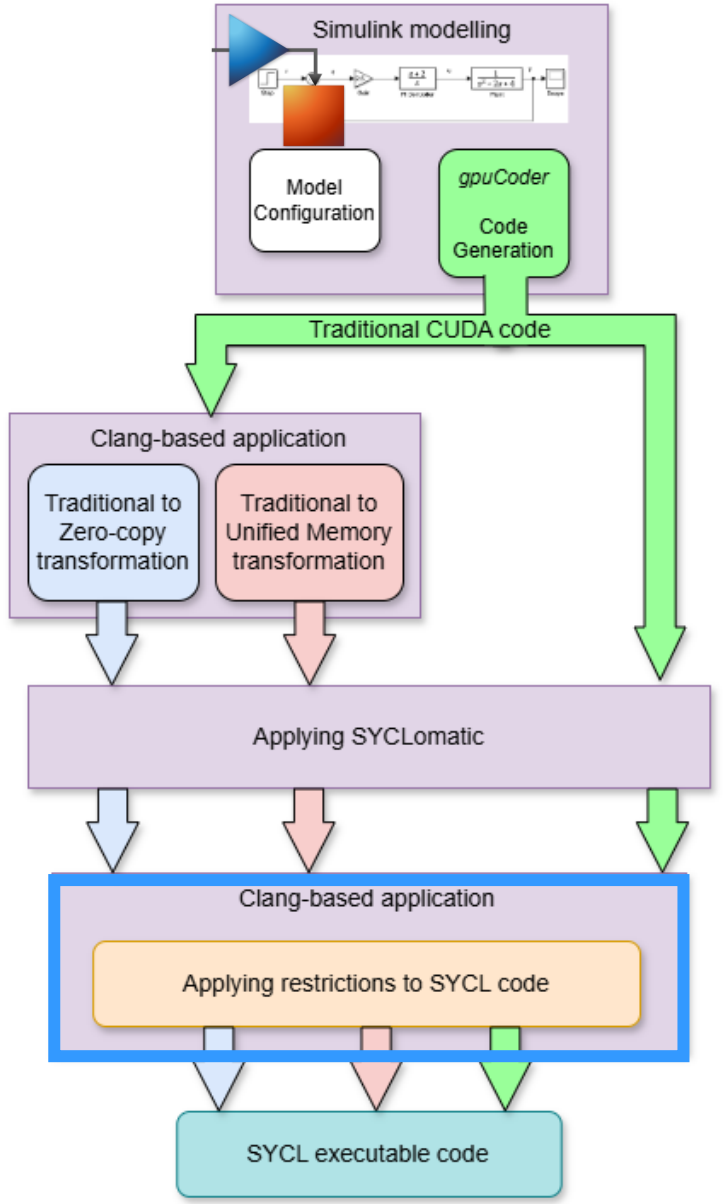
Workflow properties

- Analysability
- Use of high-level language
- Easy to learn
- Verifiability & Testability
- Traceability
- Bounded memory behavior
- Portability / HW abstraction
- Avoidance of unsafe language constructs



Workflow properties

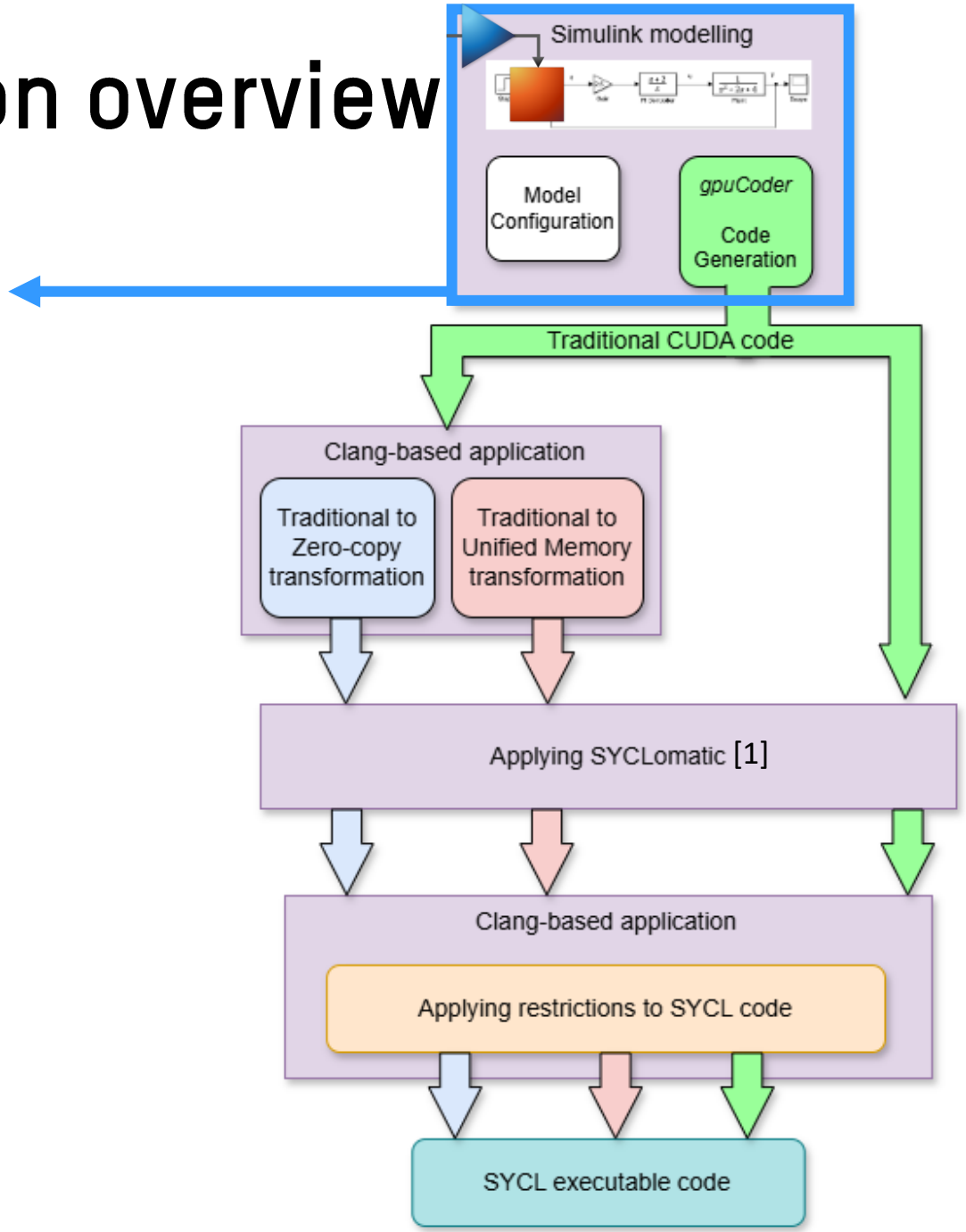
- Analysability
- Use of high-level language
- Easy to learn
- Verifiability & Testability
- Traceability
- Bounded memory behavior
- Portability / HW abstraction
- Avoidance of unsafe language constructs



3. Proposed approach: Implementation.

Implementation overview

Configuration tweaks to improve generated code produced



Simulink model tweaks

coder.inline('always')
This ensures that the generated code is emitted as a single consolidated function, rather than being decomposed into multiple auxiliary functions

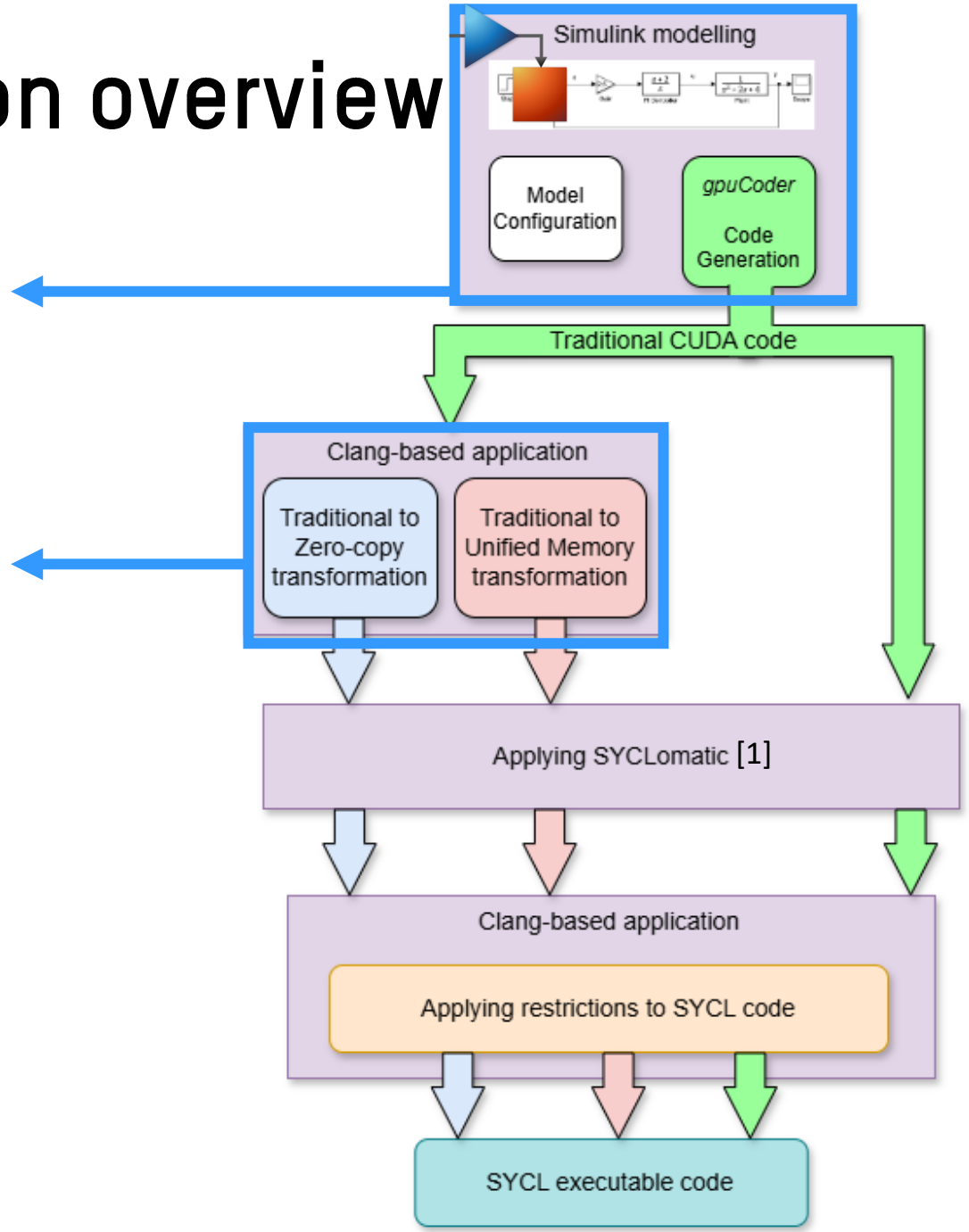
AVOID emxArray-based representations
Arrays are instead represented using statically sized data structures, eliminating dynamic memory allocation at runtime

```
coder.inline('always');

gray = coder.nullcopy(zeros(H, W, 'single'));
sobelX = coder.nullcopy(zeros(H, W, 'single'));
sobelY = coder.nullcopy(zeros(H, W, 'single'));
x_binary = coder.nullcopy(false(H, W));
y_binary = coder.nullcopy(false(H, W));
gradMag_sq = coder.nullcopy(zeros(H, W, 'single'));
mag_binary = coder.nullcopy(false(H, W));
dirAngle = coder.nullcopy(zeros(H, W, 'double'));
dir_binary = coder.nullcopy(false(H, W));
s_binary = coder.nullcopy(false(H, W));
v_binary = coder.nullcopy(false(H, W));
color_binary = coder.nullcopy(false(H, W));
final_binary = coder.nullcopy(false(H, W));
```

Implementation overview

Configuration tweaks to improve generated code produced

Clang application to change traditional memory model



Clang application to modify CUDA memory models

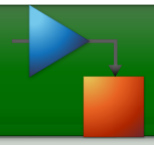
CUDA memory models features

Property	Traditional	Zero-copy	Unified Memory
Data movement	Explicit	None (remote access)	Implicit (migration)
Memory location	GPU	Host	Dynamic (CPU/GPU)
Control	High	Medium	Low
Predictability	Medium	High	Medium
Complexity	High	Medium	Low
Safety suitability	Low	High	Medium

- All memory allocation methods are affected by GPU architecture and workload characteristics.
- Other studies showed traditional allocation is often faster but more variable, while zero-copy and unified memory may or may no improve execution time but usually yield more consistent performance (lower standard deviation).

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

Clang application to modify CUDA memory models



Traditional to Zero-copy

- Detect memory transfers (*cudaMemcpy*)
- Infer data-flow between host/device
- Replace cudaMalloc → *cudaHostAllocMapped*
- Insert mapped device pointers → *cudaHostGetDevicePointer*
- Remove redundant copies

```
sobel_prueba_7_checkCudaError_g(cudaMalloc(&gpu_pln3, sizeof
(*gpu_pln3)), __FILE__, __LINE__);

sobel_prueba_7_checkCudaError(cudaMemcpy(*gpu_pln3,
sobel_prueba_7_DW.cpu_pln3, 921600UL, cudaMemcpyHostToDevice),
__FILE__, __LINE__);

sobel_prueba_7_checkCudaError_g(cudaFree(gpu_pln3), __FILE__,
__LINE__);
```

```
sobel_prueba_7_checkCudaError(cudaHostAlloc((void
**)&sobel_prueba_7_DW.cpu_pln3, 921600UL, cudaHostAllocMapped),
__FILE__, __LINE__);

sobel_prueba_7_checkCudaError(cudaHostGetDevicePointer((void
**)&gpu_pln3, (void *)sobel_prueba_7_DW.cpu_pln3, 0), __FILE__,
__LINE__);

sobel_prueba_7_checkCudaError(cudaFreeHost(sobel_prueba_7_DW.cp
u_pln3), __FILE__, __LINE__);
```

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

Clang application to modify CUDA memory models

Traditional to Unified Memory

- Detect data transfers (*cudaMemcpy*)
- Infer host–device relationships
- Replace cudaMalloc → *cudaMallocManaged*
- Remove explicit copies

```
sobel_prueba_7_checkCudaError_g(cudaMalloc(&gpu_pln3, sizeof
(*gpu_pln3)), __FILE__, __LINE__);

sobel_prueba_7_checkCudaError(cudaMemcpy(*gpu_pln3,
sobel_prueba_7_DW.cpu_pln3, 921600UL, cudaMemcpyHostToDevice),
__FILE__, __LINE__);

sobel_prueba_7_checkCudaError_g(cudaFree(gpu_pln3), __FILE__,
__LINE__);
```

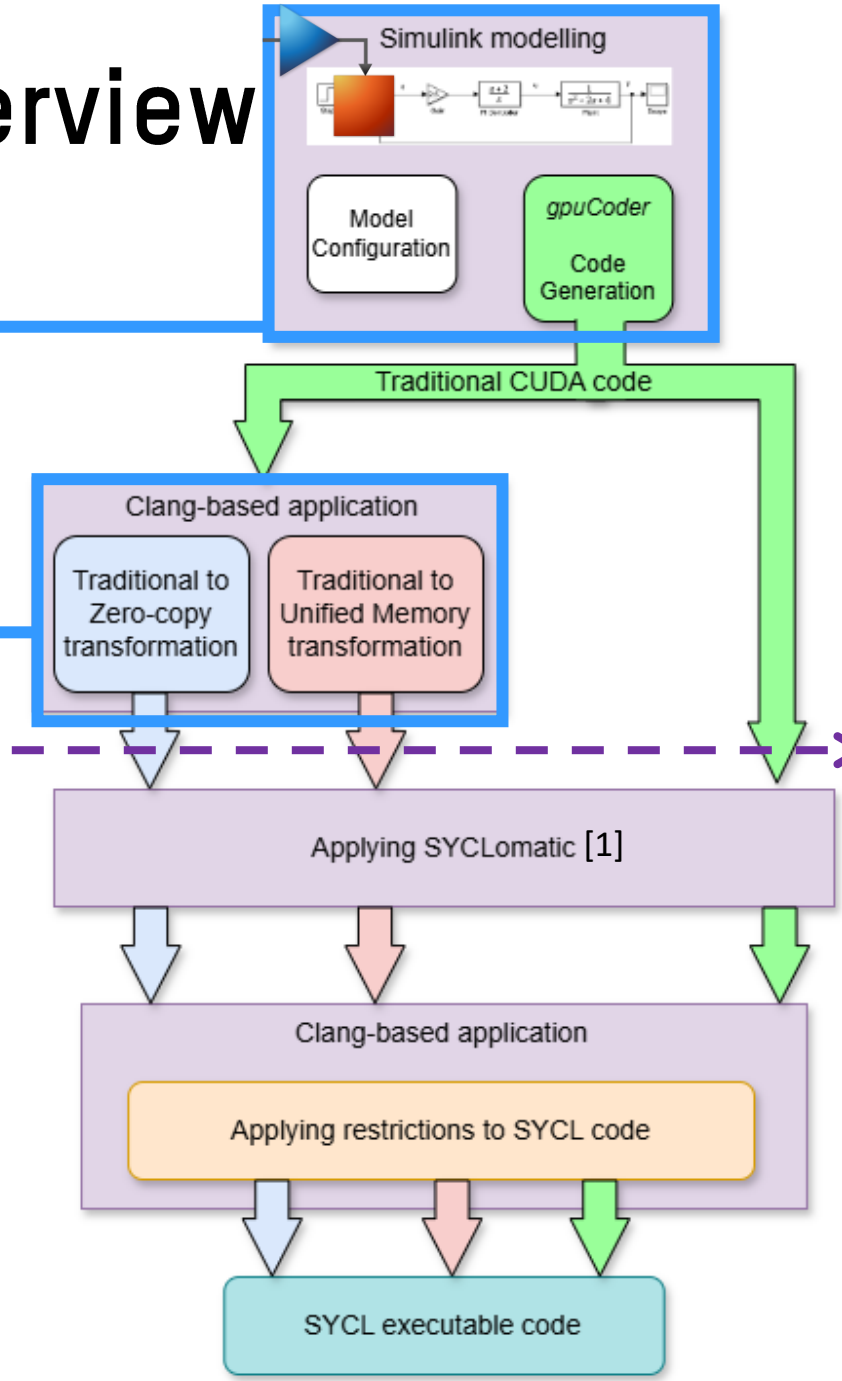
```
sobel_prueba_7_checkCudaError(cudaMallocManaged((void
**)&sobel_prueba_7_DW.cpu_pln3, 921600UL), __FILE__, __LINE__);


sobel_prueba_7_checkCudaError(cudaFree(sobel_prueba_7_DW.cpu_pl
n3), __FILE__, __LINE__);
```

ikerlan MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

Implementation overview

Configuration tweaks to improve generated code produced

Clang application to change traditional memory model

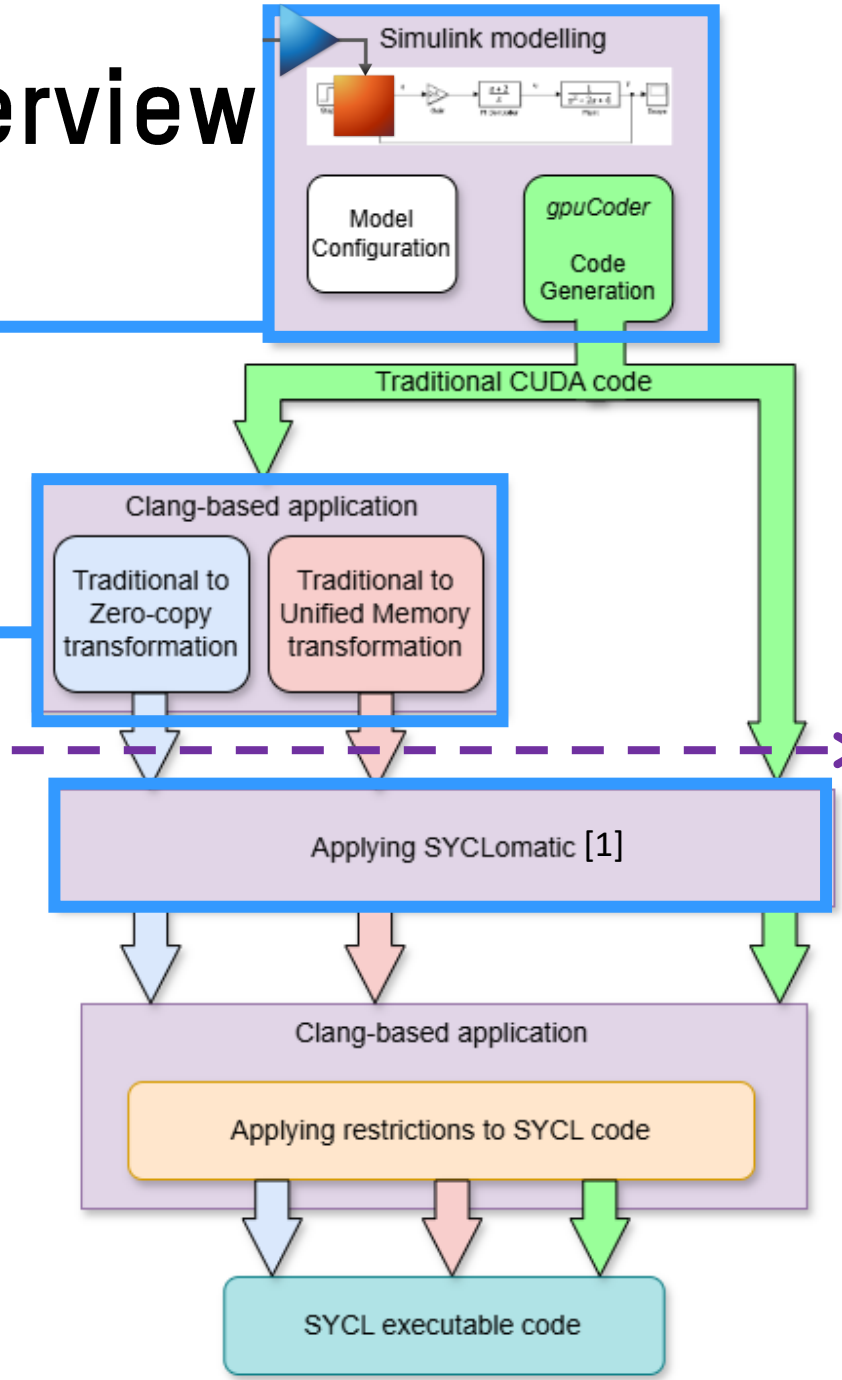


CUDA code using three different memory management models

Implementation overview

Configuration tweaks to improve generated code produced

Clang application to change traditional memory model

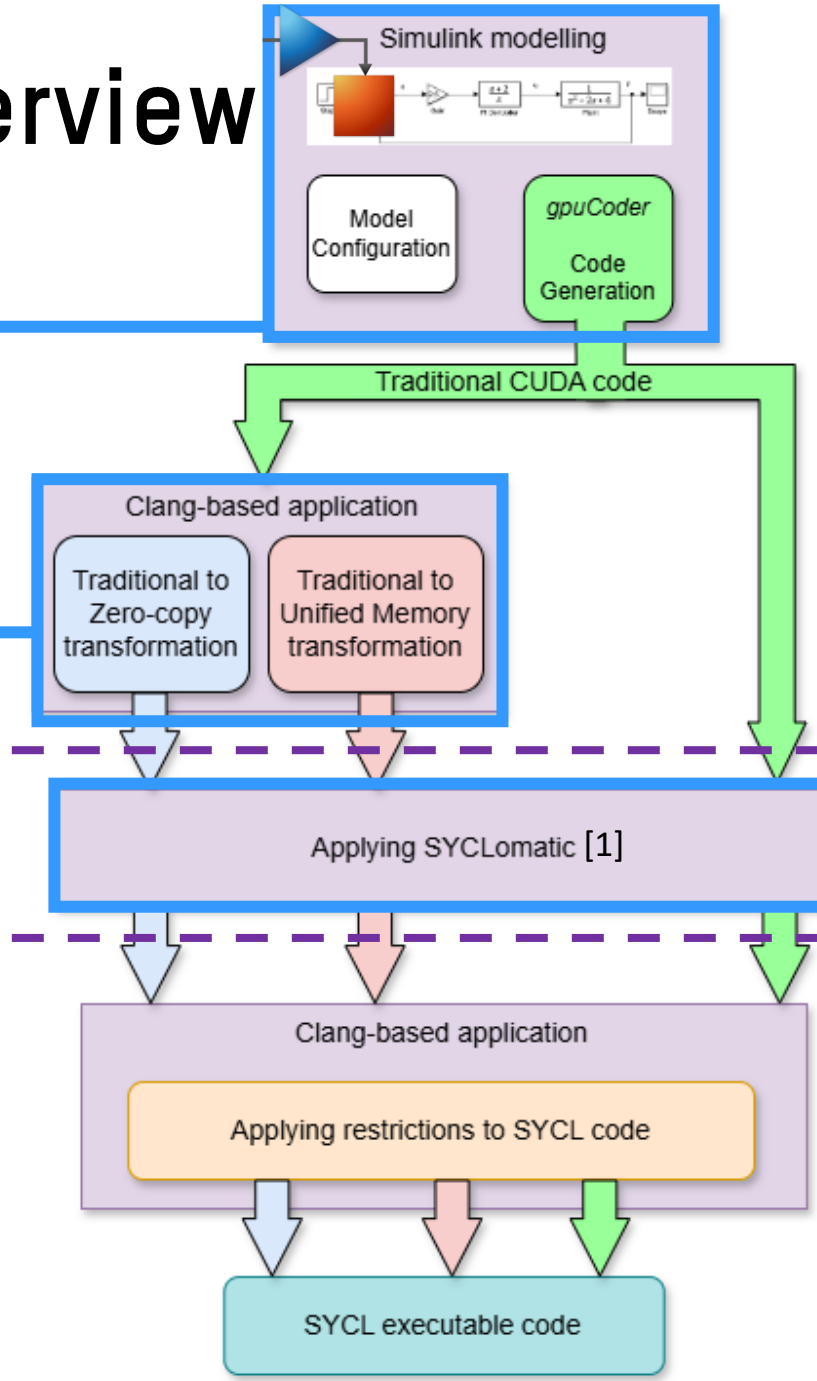


CUDA code using three different memory management models

Implementation overview

Configuration tweaks to improve generated code produced

Clang application to change traditional memory model



CUDA code using three different memory management models

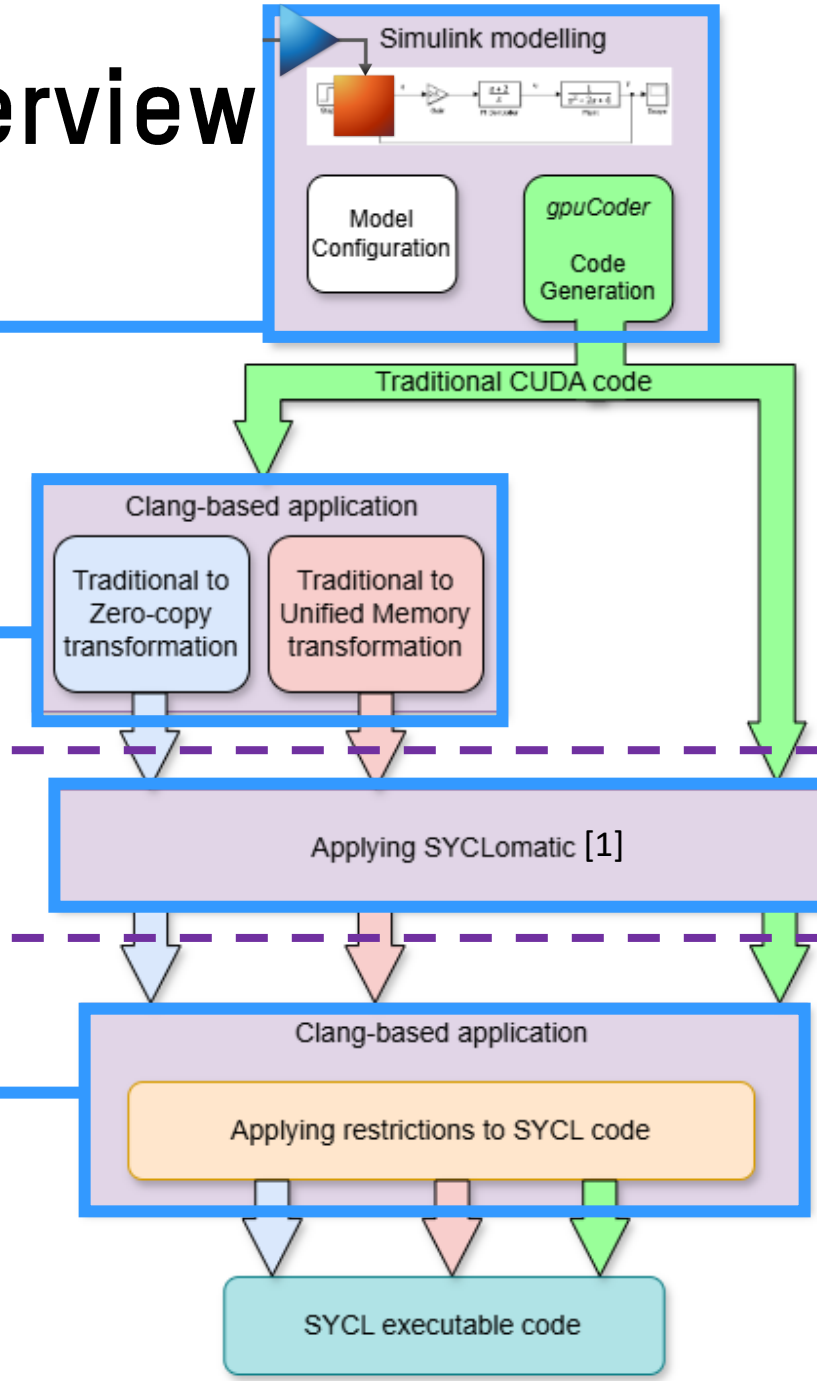
SYCL code using three different memory management models

Implementation overview

Configuration tweaks to improve generated code produced

Clang application to change traditional memory model

Clang application to apply restrictions to SYCL generated code



CUDA code using three different memory management models

SYCL code using three different memory management models

Restrictions applied by the second clang application over SYCLomatic output

The transformations are explicitly motivated by recommendations from **ISO 26262-6:2018** [2]

- ISO 26262-6:2018, Table 1, Recommendation 1b) *recommendation on the exclusion of language constructs that may result in unhandled run-time errors*
- ISO 26262-6:2018, Table 1, Recommendation 1e) *supports the use of consistent naming and structural conventions*
- ISO 26262-6:2018, Table 1, Recommendation 1h) *dynamic allocation introduces nondeterministic behaviour due to runtime memory management and potential fragmentation.*
- ISO 26262-6:2018, Table 3, Recommendation 1i) *recommendation for appropriate management of shared resources*
- General requirement for controlled concurrency and explicit synchronization mechanisms, as encouraged throughout ISO 26262-6 to *avoid unintended timing and data hazards*

Restrictions applied by the second clang application over SYCLomatic output

The transformations are explicitly motivated by recommendations from **ISO 26262-6:2018** [2]

Migration from buffer–accessor model to USM when possible

- Provides explicit memory ownership and access patterns.
- Improves predictability and reduces implicit data movement.

```
gpu_tmp = (real_T(*)[9])sycl::malloc_device(72UL, q_ct1);
```

```
sycl::multi_ptr<real_T, sycl::access::address_space::global_space>
gpu_tmp(sycl::malloc_device<real_T>(72UL, q_ct1));
```

```
 q_ct1.submit([&](sycl::handler &cgh) {
    uint8_T *gpu_rtb_imag_ct0 = *gpu_rtb_imag;

    cgh.parallel_for(sycl::nd_range<3>(sycl::range<3>(1U,
1U, 5400U) * sycl::range<3>(1U, 1U, 512U),
sycl::range<3>(1U, 1U, 512U)), [=](sycl::nd_item<3>
item_ct1) {
 sobel_prueba_7_Outputs_kernel1(gpu_rtb_imag_ct0, item_ct1);
});
  });
  q_ct1.wait();
```

```
q_ct1.submit([&](sycl::handler& cgh) {
      uint8_T* gpu_rtb_imag_ct0 = gpu_rtb_imag.get();

      cgh.parallel_for(sycl::nd_range<3>(sycl::range<3>(1U,
1U, 5400U) * sycl::range<3>(1U, 1U,512U), sycl::range<3>(1U,
1U, 512U)), [=](sycl::nd_item<3> item_ct1) {
              sobel_prueba_7_Outputs_kernel1(gpu_rtb_imag_ct
0, item_ct1);
          });
  });
  q_ct1.wait();
```

Restrictions applied by the second clang application over SYCLomatic output

The transformations are explicitly motivated by recommendations from **ISO 26262-6:2018** [2]

Migration from buffer–accessor model to USM when possible

- Provides explicit memory ownership and access patterns.
- Improves predictability and reduces implicit data movement.

Strongly typed memory management (USMBuffer)

- Couples pointer + size into a single semantic unit.
- Enables bounded access, traceability, and safer resource usage.

```
template<typename T>
struct USMBuffer {
    T* ptr;
    std::size_t size;
};
```

```
USMBuffer<int32_T> cpu_b {
sycl::malloc_shared<int32_T>(2880UL, q_ct1), 2880UL };

USMBuffer<real_T> gpu_tmp {
sycl::malloc_device<real_T>(72UL, q_ct1), 72UL };
```

Restrictions applied by the second clang application over SYCLomatic output

The transformations are explicitly motivated by recommendations from **ISO 26262-6:2018** [2]

Migration from buffer–accessor model to USM when possible

- Provides explicit memory ownership and access patterns.
- Improves predictability and reduces implicit data movement.

Strongly typed memory management (USMBuffer)

- Couples pointer + size into a single semantic unit.
- Enables bounded access, traceability, and safer resource usage.

Kernel explicit synchronization

- Enforces barriers to prevent race conditions.
- Ensures well-defined execution order across kernels.

Restrictions applied by the second clang application over SYCLomatic output

The transformations are explicitly motivated by recommendations from **ISO 26262-6:2018** [2]

Migration from buffer–accessor model to USM when possible

- Provides explicit memory ownership and access patterns.
- Improves predictability and reduces implicit data movement.

Strongly typed memory management (USMBuffer)

- Couples pointer + size into a single semantic unit.
- Enables bounded access, traceability, and safer resource usage.

Kernel explicit synchronization

- Enforces barriers to prevent race conditions.
- Ensures well-defined execution order across kernels.

Elimination of dpct dependencies

- Removes opaque tool-generated abstractions.
- Improves portability, transparency, and certifiability.

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE
UPC
BSC

Restrictions applied by the second clang application over SYCLomatic output

The transformations are explicitly motivated by recommendations from **ISO 26262-6:2018** [2]

Migration from buffer–accessor model to USM when possible

- Provides explicit memory ownership and access patterns.
- Improves predictability and reduces implicit data movement.

Strongly typed memory management (USMBuffer)

- Couples pointer + size into a single semantic unit.
- Enables bounded access, traceability, and safer resource usage.

Kernel explicit synchronization

- Enforces barriers to prevent race conditions.
- Ensures well-defined execution order across kernels.

Elimination of dpct dependencies

- Removes opaque tool-generated abstractions.
- Improves portability, transparency, and certifiability.

Removal of automatic error handling (dpct)

- Eliminates implicit control flows (exceptions, callbacks).
- Improving analysability

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE
UPC
BSC

Restrictions applied by the second clang application over SYCLomatic output

The transformations are explicitly motivated by recommendations from **ISO 26262-6:2018** [2]

Migration from buffer–accessor model to USM when possible

- Provides explicit memory ownership and access patterns.
- Improves predictability and reduces implicit data movement.

Strongly typed memory management (USMBuffer)

- Couples pointer + size into a single semantic unit.
- Enables bounded access, traceability, and safer resource usage.

Kernel explicit synchronization

- Enforces barriers to prevent race conditions.
- Ensures well-defined execution order across kernels.

Elimination of dpct dependencies

- Removes opaque tool-generated abstractions.
- Improves portability, transparency, and certifiability.

Removal of automatic error handling (dpct)

- Eliminates implicit control flows (exceptions, callbacks).
- Improving analysability

Refactoring queue management to native SYCL

- Uses only standard SYCL constructs.

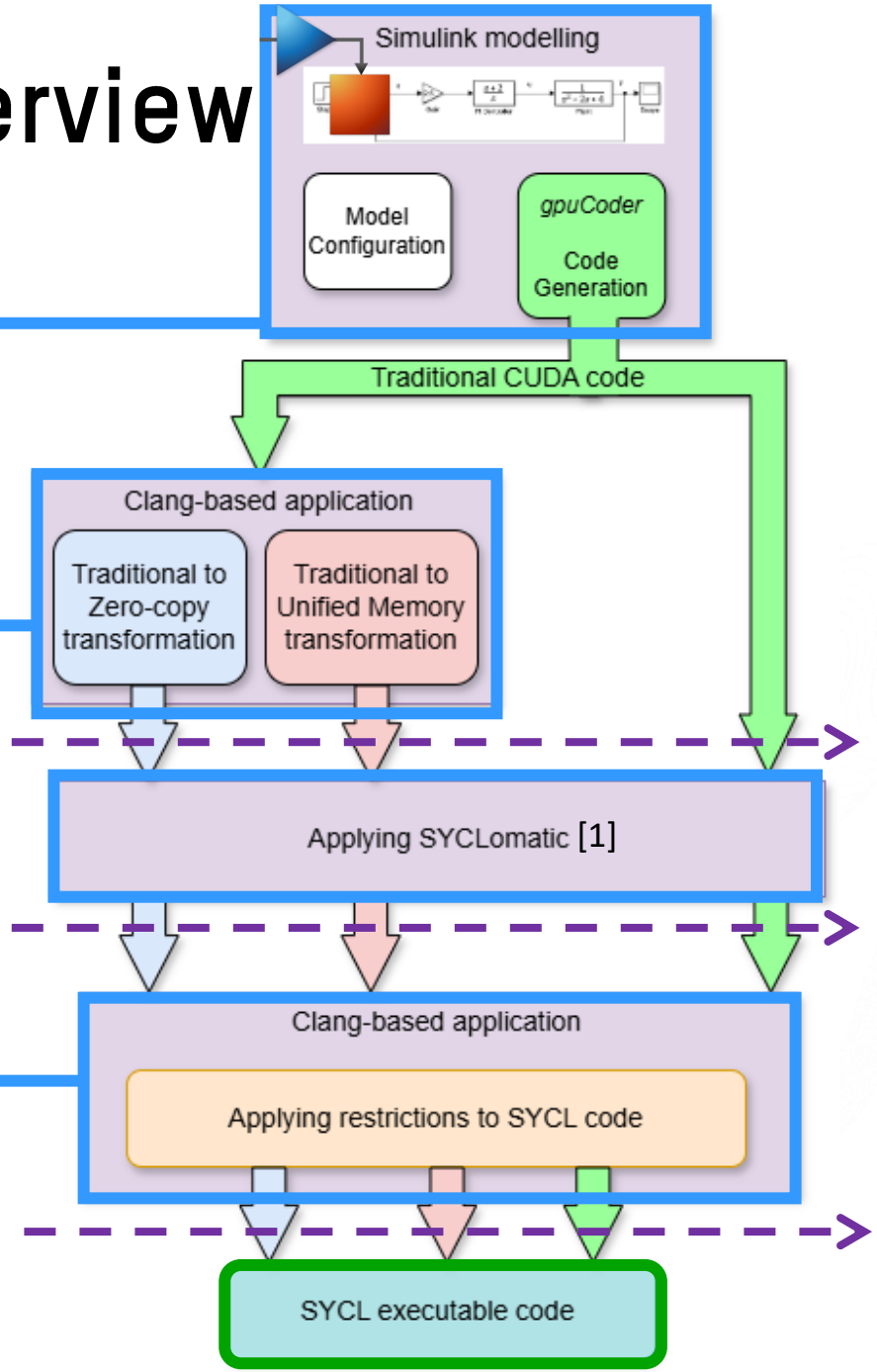
ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

Implementation overview

Configuration tweaks to improve generated code produced

Clang application to change traditional memory model

Clang application to apply restrictions to SYCL generated code



CUDA code using three different memory management models

SYCL code using three different memory management models

SYCL code using three different memory management models under safety-compliant restrictions

4. Experimental setup:
Use case

Hardware used

Desktop Computer

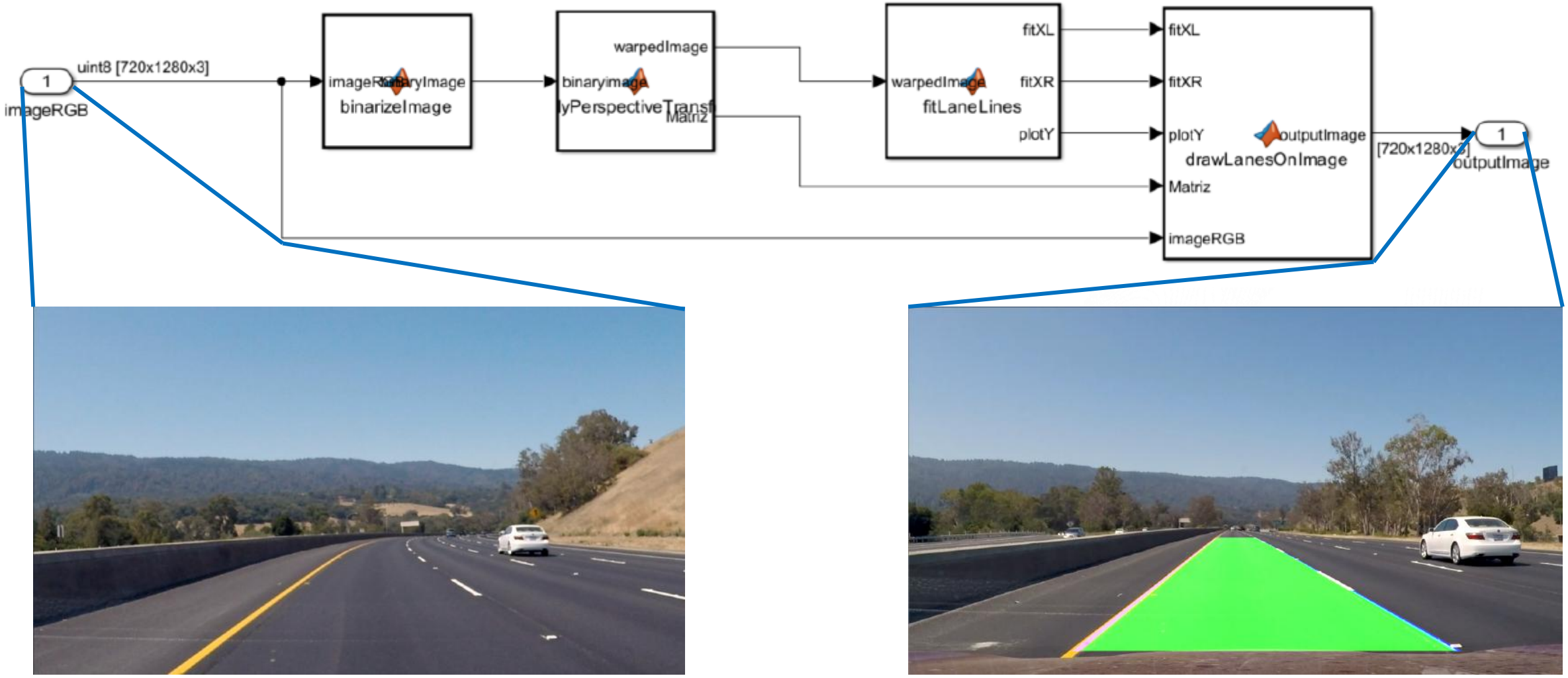
- AMD Ryzen 9
- NVIDIA RTX 4090 GPU (Ada Lovelace) - 16,384 CUDA cores
- Compiler: Intel oneAPI DPC++/C++ Compiler version 2025.1.0

NVIDIA Jetson Orin AGX

- Arm Cortex-A78AE (v8.2, 64-bit)
- GA10B (Ampere) - 2048 CUDA cores
- 50 W profile
- Compiler: AdaptiveCpp version based on LLVM/Clang 19.1.7.

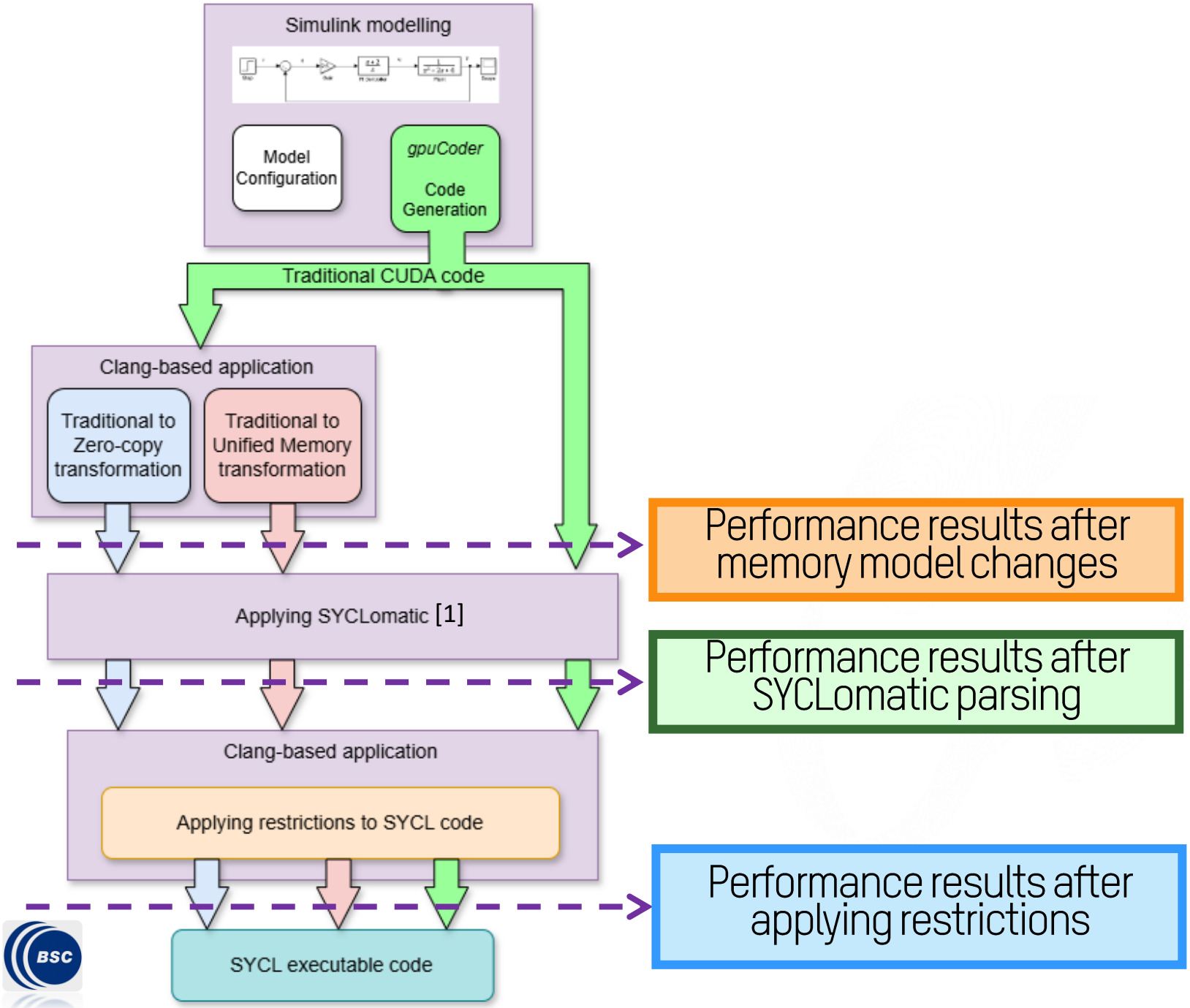
ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

Use case model [3]

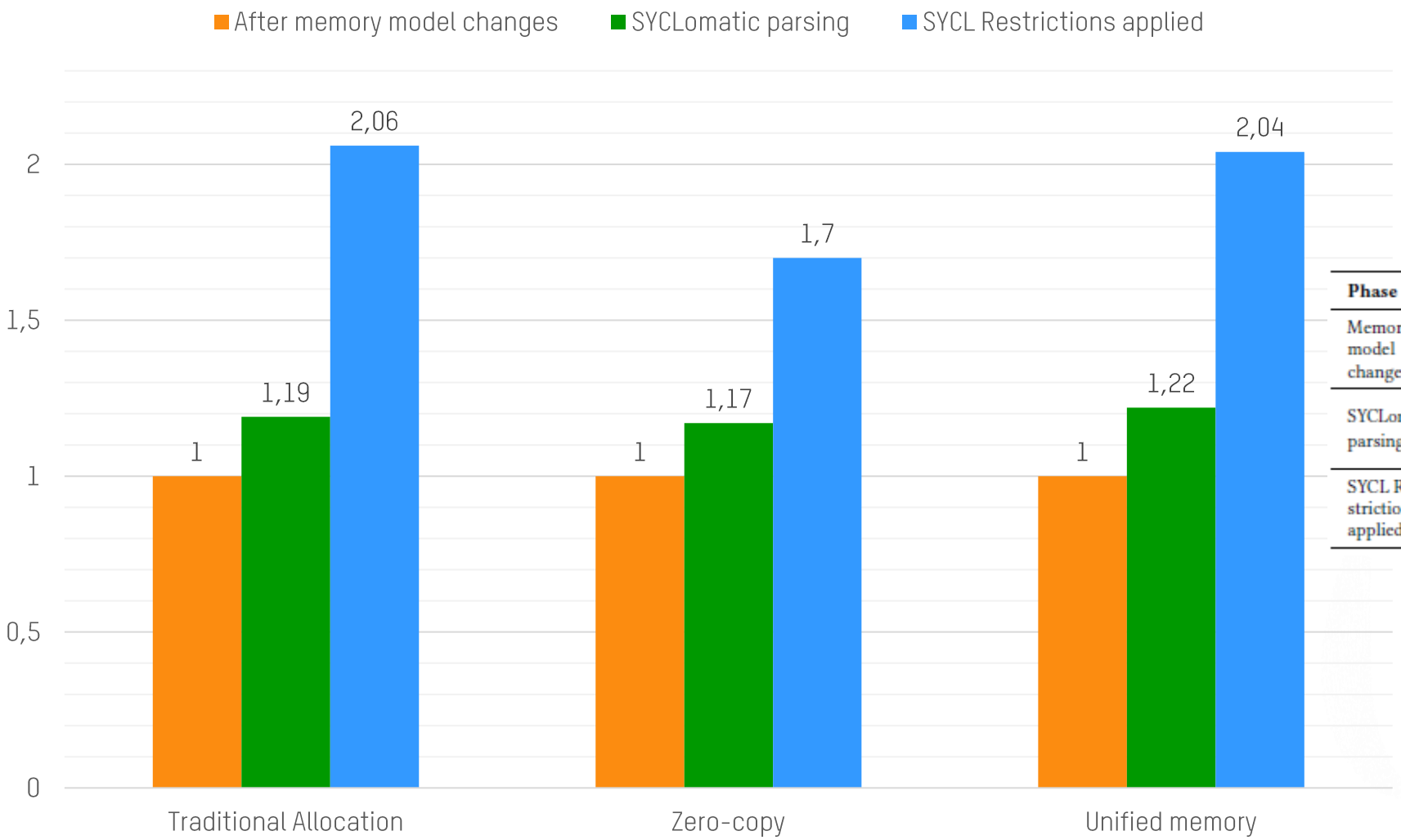


5. Results:
Data extracted

Results


Simulink modelling
Model Configuration
gpuCoder
Code Generation
Traditional CUDA code
Clang-based application
Traditional to Zero-copy transformation
Traditional to Unified Memory transformation
Applying SYCLomatic [1]
Clang-based application
Applying restrictions to SYCL code
SYCL executable code
Performance results after memory model changes
Performance results after SYCLomatic parsing
Performance results after applying restrictions

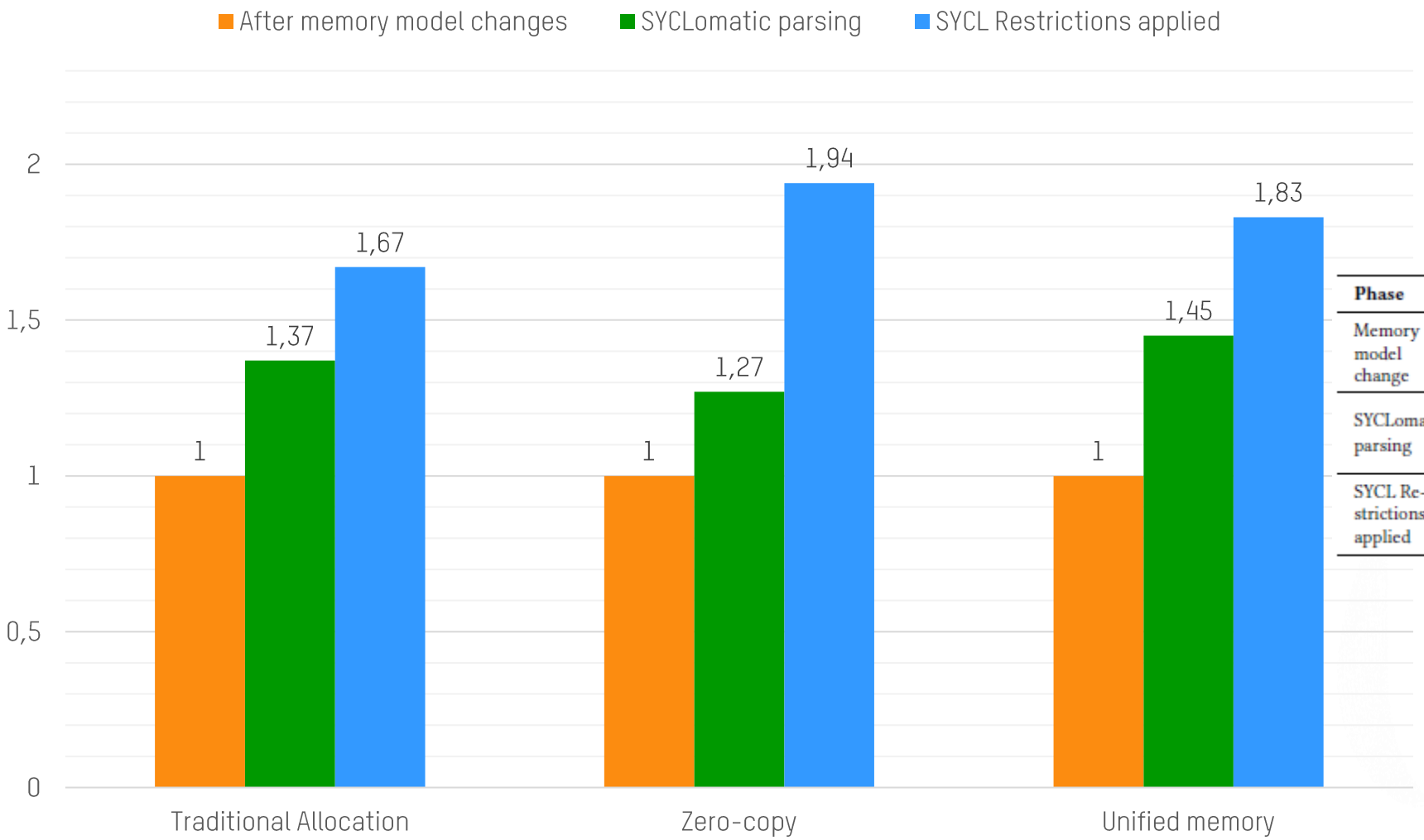
AVERAGE STEP TIME (NORMALISED) ON THE DESKTOP PLATFORM



Phase	Memory Model	Step Time (ms)	Step ratio	Avg FPS
Memory model change	CUDA Traditional	10.986 ± 0.474	x1	89.240 ± 2.466
	CUDA Zero-Copy	17.570 ± 0.348	x1	56.327 ± 0.815
	CUDA Unified Memory	11.994 ± 0.293	x1	81.722 ± 1.899
SYCLomatic parsing	SYCL Traditional	13.068 ± 0.200	x1.19	76.094 ± 0.258
	SYCL Zero-Copy	20.543 ± 0.336	x1.17	48.642 ± 0.127
	SYCL Unified Memory	14.609 ± 0.302	x1.22	67.430 ± 0.176
SYCL Restrictions applied	SYCL Traditional	22.585 ± 0.150	x2.06	44.004 ± 0.107
	SYCL Zero-Copy	29.836 ± 0.351	x1.70	33.506 ± 0.071
	SYCL Unified Memory	24.452 ± 0.277	x2.04	40.842 ± 0.030

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE
UPC
BSC

AVERAGE STEP TIME (NORMALISED) ON THE JETSON ORIN AGX PLATFORM



Phase	Memory Model	Step Time (ms)	Step ratio	Avg FPS
Memory model change	CUDA Traditional	97.177 ± 6.522	x1	10.850 ± 0.839
	CUDA Zero-Copy	96.475 ± 3.101	x1	10.389 ± 0.104
	CUDA Unified Memory	100.940 ± 1.765	x1	10.166 ± 0.130
SYCLomatic parsing	SYCL Traditional	133.287 ± 2.507	x1.37	6.157 ± 1.017
	SYCL Zero-Copy	122.508 ± 1.631	x1.27	8.171 ± 0.020
	SYCL Unified Memory	146.394 ± 3.007	x1.45	7.474 ± 0.310
SYCL Restrictions applied	SYCL Traditional	162.364 ± 18.827	x1.67	6.597 ± 0.295
	SYCL Zero-Copy	186.947 ± 3.672	x1.94	5.374 ± 0.018
	SYCL Unified Memory	184.701 ± 2.763	x1.83	5.360 ± 0.063

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

6. Conclusions

Key Contributions

- ✓ End-to-end workflow from **Simulink models → GPU implementation**
- ✓ Founded on **Model-Based Design (MBD)** for early validation & verification
- ✓ Combines:
  - Automatic CUDA generation
  - Memory restructuring
  - SYCL post-processing for safety

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

Limitations & Future work

- ❗ Not a fully compliant safety process (ISO 26262 lifecycle not complete)

- ❗ Tooling limitations (AdaptiveCpp): incomplete **USM support**

- Promising **pre-certification workflow**

- Requires further work in:
  - Embedded optimization
  - Toolchain maturity

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

References

[1] Robert Mueller-Albrecht. 2024. SYCLomatic: SYCL Adoption for Everyone - Moving from CUDA to SYCL Gets Progressively Easier: Advanced Migration Considerations. In Proceedings of the 12th International Workshop on OpenCL and SYCL (Chicago, IL, USA) (IWOCL '24)

[2] International Organization for Standardization. (2018). ISO 26262-6:2018 Road vehicles — Functional safety — Part 6: Product development at the software level. ISO.

[3] Sri Anumakonda. 2021. Pairing Lane Detection with Object Detection. https://srianumakonda.medium.com/pairing-lane-detection-with-object-detection-665b30462952 -

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

Thank you for your attention!

Marcos Rodriguez
Universitat Politècnica de Catalunya (UPC), Spain
Ikerlan Technology Research Center, Spain

Jon Pedernales
Universitat Politècnica de Catalunya (UPC), Spain
Ikerlan Technology Research Center, Spain

Leonidas Kosmidis
Barcelona Super Computing Center (BSC), Spain
Universitat Politècnica de Catalunya (UPC), Spain

Alejandro Calderón
Ikerlan Technology Research Center, Spain

Irune Yarza
Ikerlan Technology Research Center, Spain