LibreRT: Portable Heterogeneous Real-Time Programming for the Embedded Computing Continuum

Invited Paper

Biagio Cosenza
University of Salerno
Fisciano (Salerno), Italy
bcosenza@unisa.it

Federico Terraneo
Politecnico di Milano
Milano, Italy
federico.terraneo@polimi.it

Giovanni Agosta
Politecnico di Milano
Milano, Italy
giovanni.agosta@polimi.it

Lorenzo Carpentieri
University of Salerno
Fisciano (Salerno), Italy
lcarpentieri@unisa.it

Daniele Cattaneo
Politecnico di Milano
Milano, Italy
daniele.cattaneo@polimi.it

Mohammad VazirPanah
University of Salerno
Fisciano (Salerno), Italy
mvazirpanah@unisa.it

Niccolò Nicolosi
Politecnico di Milano
Milano, Italy
niccolo.nicolosi@polimi.it

Abstract

The LibreRT project aims to address the lack of portable system software that can efficiently target the full spectrum of embedded devices, ranging from ultra-low-power micro-edge micro-controllers to deep- and meta-edge systems with embedded GPUs. It is based on three technological pillars: (1) Portable programming models and OS support for highly responsive real-time processing on micro-edge devices; (2) Portable programming for high-throughput real-time processing on heterogeneous deep- and meta-edge systems; (3) Real-time processing leveraging approximate computing and autotuning. LibreRT proposes a number of new technology advances: the programming model, based on the SYCL standard, has been extended with two novel extensions: SYrtos, which introduces priority- and deadline-based scheduling into SYCL with preemptive multitasking on OS integration; and SYprox, which provides approximate computing techniques integration into SYCL language semantics. The LibreRT software stack integrates programming models and compiler with complementary operating system technologies. Specifically, LibreRT proposes a new operating system kernel architecture called fluid kernel, designed for embedded system support, and a multi-core real-time scheduler for micro-edge devices and a high-resolution timing subsystem. Both features have been implemented in the Miosix OS. Overall, the LibreRT software stack has been validated on several benchmarks, including AI and image processing applications, and demonstrated in a real-world real-time scenario using a thermal camera prototype.

CCS Concepts

• **Software and its engineering** → **Parallel programming languages**; • **Computer systems organization** → **Real-time languages**; *Embedded systems.*

This work is licensed under a Creative Commons Attribution 4.0 International License.

CF Companion '26, Catania, Italy
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2569-2/26/05
https://doi.org/10.1145/3801488.3806240

Keywords

SYCL, real-time systems, embedded GPUs, approximate computing, operating systems, programming models

ACM Reference Format:
Biagio Cosenza, Federico Terraneo, Giovanni Agosta, Lorenzo Carpentieri, Daniele Cattaneo, Mohammad VazirPanah, and Niccolò Nicolosi. 2026. LibreRT: Portable Heterogeneous Real-Time Programming for the Embedded Computing Continuum: Invited Paper. In *Proceedings of the 23rd ACM International Conference on Computing Frontiers (CF Companion '26), May 19–21, 2026, Catania, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3801488.3806240

1 Introduction and Motivations

The emerging paradigms of *Industrial IoT* (IIoT) and edge computing, combined with the increasing interest in applications such as machine learning and image processing, have drastically increased the demand for embedded systems that are, on one hand, highly responsive and capable of handling real-time requirements, and, on the other hand, can deliver high computational power. Heterogeneous embedded systems with low-power GPUs have become ubiquitous and can be found in domains ranging from wearable devices and mobile computing to automotive systems. With such *extreme heterogeneity* [25] on the horizon, heterogeneous architectures are set to pervade embedded systems, from ultra-low-power micro-edge devices to the more demanding deep- and meta-edge systems of the automotive domain.

However, there are no portable system software solutions in terms of programming models, compilers, and operating systems, that can efficiently target all devices in the embedded systems computing continuum.

From the programming model perspective, C and C++ remain the most popular languages for embedded systems. Embedded systems programming remains fragmented: while mature C APIs enable real-time programmability for embedded CPUs, no equivalent solution exists for heterogeneous systems incorporating embedded GPUs. SYCL [20] is an emerging programming model based on modern C++ for heterogeneous parallel programming; it is an open, royalty-free standard by the industry consortium Khronos Group and, in its latest version SYCL 2020, supports multiple back-ends in

modern C++17. SYCL has been extended in several ways to provide advanced features and address a variety of use cases. Two notable examples include Celerity [23], which targets programming clusters of accelerators by providing an asynchronous dataflow-driven execution model [17] as well as support for scalability prediction [19], and SYnergy [12], which focuses on energy-efficient computing with energy modeling [5] and fine-grained phase-based frequency scaling [6]. However, no existing extension currently addresses real-time for embedded systems.

On the operating system side, existing embedded real-time *operating systems* (OSes) such as FreeRTOS and Mbed provide limited or no POSIX compliance and restricted C/C++ standard library support, increasing programming complexity and preventing code sharing between embedded and non-embedded environments. Real-time OSes often expose limitations that prevent adoption along the continuum — for example, limited support for high-resolution timing as well as costly handling of kernel and user space transitions.

The LibreRT project [24] aims at providing a portable system software stack that is capable of delivering highly responsive, real-time, and high-throughput computation for the embedded computing continuum. Specifically, LibreRT proposes a SYCL-based programming model with extensions for real-time as well as approximate computing for higher performance. The LibreRT programming model builds on modern C++ semantics and advanced runtime systems, and is capable of targeting embedded GPUs. For the operating system, LibreRT targets micro-edge devices with Miosix [3], an OS with higher responsiveness and advanced scheduling support, and deep- and meta-edge devices with more traditional Linux distributions. Finally, an autotuning layer introduces an adaptive tuning methodology that balance approximation accuracy with performance to meet the real-time constraints.

2 LibreRT Project Overview

The LibreRT project addresses a critical gap in the embedded systems landscape: the lack of portable system software solutions—in terms of programming models, compilers, and operating systems—that can efficiently target the full spectrum of heterogeneous embedded devices. This spectrum ranges from ultra-low-power micro-edge micro-controllers to more powerful deep- and meta-edge systems equipped with embedded GPUs, which are increasingly deployed in domains such as automotive, industrial IoT, and edge computing.

LibreRT delivers a portable software stack capable of supporting highly responsive, real-time, and high-throughput computation across the embedded computing continuum.

2.1 Objectives

The LibreRT project is organized around four concrete objectives.

> *O1: Portable programming models and OS for high-responsive real-time processing on micro-edge devices.*

This objective focuses on extending the SYCL programming model with real-time semantics (priorities and deadlines) and mapping these extensions onto Miosix's advanced scheduling features for micro-edge CPUs.

> *O2: Portable programming for high-throughput real-time processing on heterogeneous deep- and meta-edge devices.*

This objective targets devices equipped with embedded GPUs (e.g., ARM Mali, NVIDIA Jetson), providing SYCL-based kernel offloading with real-time scheduling support through a software GPU scheduler implementing priorities and preemption.

> *O3: Technologies for real-time processing enabled by approximate computing and autotuning.*

This objective develops approximation techniques (mixed precision and kernel perforation) integrated into SYCL, and autotuning strategies that dynamically select the best configuration to meet real-time constraints while maximizing accuracy.

> *O4: Validation on representative embedded benchmarks.*

The project validates the developed technologies on three application domains: image processing, neural networks for high-throughput deep- and meta-edge devices, and highly responsive real-time applications for micro-edge. The target architectures of the project include three classes of embedded systems: ultra-low-power micro-edge devices based on Miosix OS, e.g., multi-core micro-controllers; deep-edge devices based on Linux with embedded GPUs such as ARM Mali; and meta-edge devices with high processing power, typically adopted in the automotive industry, such as NVIDIA Jetson platforms.

2.2 Consortium

LibreRT is a collaborative project between two Italian research units. The University of Salerno contributes expertise on SYCL programming models, SYCL extensions, and approximate computing; Politecnico di Milano contributes expertise on real-time operating systems and real-time scheduling.

3 Main Goals and Achievements

To achieve these objectives, the project develops three main technology pillars. The first pillar is the programming model based on SYCL, an open standard by Khronos Group that allows programming GPUs and accelerators with modern C++. LibreRT novelty is the extension of SYCL 2020 with novel semantics for real-time scheduling (SYrtos) and approximate computing (SYprox). The second pillar brings new innovations in terms of operating system support for real time processing. Specifically, LibreRT proposes a new operating system kernel architecture called fluid kernel. The fluid kernel architecture was designed and implemented in the Miosix OS kernel, an open-source *real time operating system* (RTOS). Miosix was also extended with a multi-core real-time scheduler for micro-edge devices and a high-resolution timing subsystem. Miosix targets the micro-controller part of the embedded computing continuum and is complemented by Linux for deep- and meta-edge devices. The third pillar is an advanced combination of technologies that accelerates computation, particularly with approximate computing techniques developed in SYprox, such as kernel perforation and mixed precision, and automatic tuning. When used alongside

real-time advances in programming models and operating systems, these techniques enable applications to meet real-time requirements in unprecedented scenarios with demanding computational needs.

3.1 Real-Time Programming

A key innovation of LibreRT is the extension of the SYCL programming model to support real-time constraints. SYCL [20], a modern C++ standard for heterogeneous parallelism, lies at the heart of the project, providing both compiler extensions to support embedded GPUs and library extensions for advanced features. SYCL has been extensively benchmarked through SYCL-Bench 2020 [11], a benchmark suite developed by the consortium to evaluate six key features of SYCL 2020 (i.e., unified shared memory, reduction kernels, specialization constants, group algorithms, in-order queues, and atomics) on GPUs from different vendors, including both high-end GPUS (from AMD, Intel and NVIDIA) as well as embedded ones (such as ARM Mali, Imagination Technologies and NVIDIA Jetson), and using both oneAPI DPC++ [2] and AdaptiveCpp [1] compilers.

LibreRT introduces **SYrtos** [9], a SYCL extension that add real-time capabilities to the language by adding support for priority and deadline to SYCL queues and kernels. The SYrtos API provides both coarse-grained priority at queue level (where all tasks submitted to a queue share the same priority) and fine-grained priority at task level (where individual kernel submissions can override the queue's default priority). The priority constants `high`, `medium`, and `low` are provided as portable definitions: on Linux, they are mapped to POSIX thread priority values, while on Miosix they map to RTOS thread priorities. The prototype implementation creates a thread pool per queue, and when multiple queues are created, each pool can operate at a different priority, enabling the OS kernel scheduler to perform preemption of lower-priority tasks.

The main entry point for using the SYrtos interface is the *queue* class, to which SYrtos adds real-time capabilities. The SYrtos API provides coarse-grained and fine-grained real-time capabilities, in particular thread priority, allowing to specify the priority of threads running on the same queue (coarse-grained) or, alternatively, only for a specific kernel execution (fine grained).

Listing 1 shows how to apply coarse-grained priority to a SYCL queue. In SYrtos, a queue may take an additional priority parameter which is applied to all tasks submitted to it. Constants `high`, `medium` and `low` are provided as convenience definitions available on all operating systems, ensuring portability. On Linux, they are mapped into priority values of 10, 5 and 1, while on Miosix to 3, 2 and 1.

```
queue q(syrtos::priority::high);
range<2> r{N,N}
buffer<float,2> inBuf(in,r);
buffer<float,2> outBuf(out,r);
q.submit([&](handler &h){
 accessor inAc(inBuf, h, read_only);
 accessor outAc(outBuf, h, write_only);
 h.parallel_for(range<2>{N, N},[&](id<2> id){
  outAc[{id[0]*2,id[1]}]=inAc[id]*2;
 });
});
```

Listing 1: Coarse-grained, queue priority

Listing 2 shows how to specify a fine-grained priority with a queue submission. In SYrtos, the queue's `submit` function supports an extra parameter that allows to specify the priority. The priority only applies within the context of that specific command group function object to the queue. If a priority has been already specified for the queue, this function overloads it.

```
queue q();
range<2> r{N,N}
buffer<float,2> inBuf(in,r);
buffer<float,2> outBuf(out,r);
q.submit([&](handler &h){
 accessor inAc(inBuf, h, read_only);
 accessor outAc(outBuf, h, write_only);
 h.parallel_for(range<2>{N, N},[&](id<2> id){
  outAc[{id[0]*2,id[1]}]=inAc[id]*2;
 });
}, syrtos::priority::high);
```

Listing 2: Fine-grained, kernel priority

SYrtos has been validated on both Intel multi-core CPUs under Linux OS and Raspberry Pi RP2040 micro-controllers under Miosix OS, demonstrating that high-priority queues consistently achieve lower and bounded latencies in the presence of thread contention. The SYrtos API also include extensions for explicit deadline support, with a real-time scheduler tailored to the SYCL programming model with task queue preemption.

The project has also investigated the portability of SYCL across the embedded computing continuum, and in particular on embedded GPUs. We evaluated all major SYCL 2020 semantics (buffer/accessor, unified shared memory, local memory mapping, and in-order queues) on a wide range of embedded GPUs including the NVIDIA Jetson Orin Nano, ARM Mali, and PowerVR IMG. To support small embedded GPUs with limited capabilities a new SYCL backend was developed, which maps buffer/accessor semantics to OpenCL's coarse-grained SVM buffer. Furthermore, the project has also investigated RISC-V backend as a target for the embedded continuum, in particular by analyzing autovectorization capabilities of GCC and LLVM compilers for the RISC-V Vector extension (RVV) on real hardware boards [7].

3.2 Operating System Achievements

LibreRT also focuses on operating system support for real-time execution, particularly on resource-constrained micro-edge devices. Two major contributions have been made in this area.

First, the concept of **fluid kernels** [21] has been introduced as a novel kernel architecture that combines the strengths of unikernels (typically found in embedded systems) with the advantages of mainstream monolithic kernels, while being capable of running efficiently and securely in MMU-less environments. Fluid kernels allow applications to seamlessly move between user space and kernel space, managing the trade-off between performance, code size, isolation, and security. The Miosix implementation of fluid kernels, compared to Linux on the same STM32 micro-controller hardware, achieves an average speedup of 3.5× (up to 15.4×) with an average code size reduction of 84% (up to 90%). Figure 2 shows the Miosix OS fluid space architecture.

Compared to FreeRTOS, the use of Miosix incurs only a moderate overhead (at most 47 KB) while providing significant performance advantages with speedups averaging 1.5× and up to 5×. The fluid kernel design supports three scheduling algorithms: priority-based, EDF, and a fair scheduling algorithm based on control theory [14].

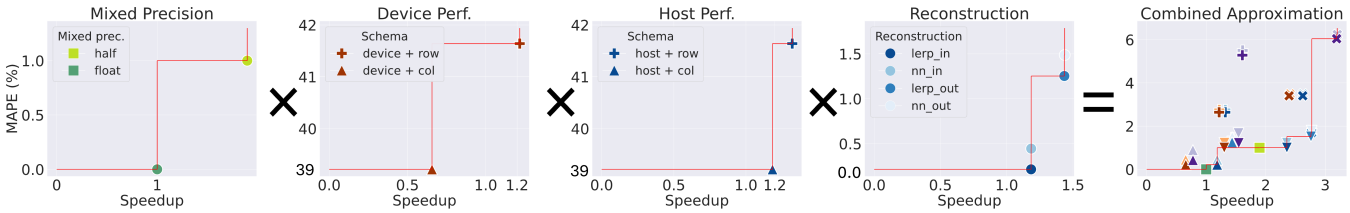

Figure 1: SYprox tuning space builds on three fundamental approximation schemes: mixed precision, device perforation and host perforation.

Second, a **high-resolution timing subsystem** [13, 22] has been designed for the Miosix OS kernel, addressing a longstanding limitation of embedded RTOSes that traditionally rely on tick-based timing with millisecond resolution. The proposed *1+N timing subsystem* design decouples preemption from the timekeeping operation by leveraging the abundance of hardware timers in modern microcontrollers. A single high-resolution timer provides microsecond-to-nanosecond resolution for timekeeping and programmed wakeups, while per-CPU timers handle preemption quantum expiration independently. This design achieves a context switch overhead boost of up to 1.75× compared to a traditional high-resolution timing implementation, bringing the overhead much closer to that of a tick-based kernel, and scales effectively to multicore micro-controller architectures.

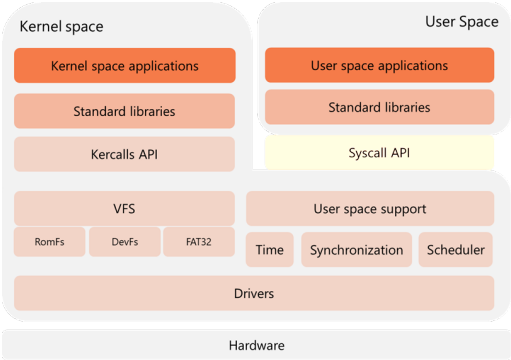

Figure 2: Miosix fluid space architecture.

3.3 Approximate Computing

To further push the performance of high-throughput embedded applications, LibreRT develops approximate computing techniques integrated in the SYCL programming model. The main contribution is **SYprox** [4], a header-only library that extends SYCL with advanced approximate computing capabilities, including data perforation, signal reconstruction, and mixed precision. SYprox makes three major breakthroughs. First, it provides a SYCL-based interface that extends the `buffer` and `accessor` classes with approximate computing capabilities (via `pbuffer` and `paccessor`). Second, it introduces *host perforation*, a novel data perforation approach that applies perforation on the host side before transferring data to the device, thereby reducing both kernel computation and data transfer times. Third, for the first time, SYprox enables the combination of data perforation and reconstruction techniques with mixed precision, expanding the approximation space with new Pareto-optimal configurations.

Listing 3 shows a SYprox code example combining host perforation with output reconstruction.

```
range<2> r = range<2>{N, N};
pbuffer<float,2,prow<float,2,nn_out>> inBuf(in,r);
q.submit([&](handler& h) {
  paccessor<float,2> inAc{inBuf, h, read_write};
  h.parallel_for(prange<>{N, N},
    [&](id<2> id) {
      outAc[id] = inAc[id] * 2;
  });
});
```

Listing 3: SYprox: host perforation with row scheme and nearest-neighbor output reconstruction.

SYprox introduces *host perforation* a novel data perforation technique implemented in SYprox. Figure 3 shows a comparison between the traditional *device perforation* and the *host perforation* approach. *Host perforation* performs data perforation on the host before sending the data to the device. In contrast, device perforation requires transferring all data from the host to the device, where they are then accessed in a perforated shape according to a pattern defined by the SYprox approximation schema. The *host perforation* offers two distinct advantages over the device perforation. Firstly, it significantly reduces the amount of data transfer needed between the host and the device. This reduction in data transfer can lead to improved performance in applications where data movement is a bottleneck. Secondly, host perforation provides a better cache utilization, since eliminates data access issues due to the data layout. By perforating the data on the host, accesses to the device data can be performed continuously, maximizing cache utilization. However, in *host perforation*, once the data have been perforated in the host, they cannot be used on the device. This limitation may affect scenarios where devices require direct access to perforated data for further processing or computations. Moreover, host perforation is only beneficial when the time required to perforate the data on the host is less than the time needed to transfer the full dataset to the device. Figure 1 shows an experimental evaluation on AMD MI100, Intel Max 1100, and NVIDIA V100 GPUs, demonstrating that SYprox's approximations are Pareto-dominant with respect to state-of-the-art approaches [15, 16, 18], and achieving up to 3.5× speedup with only 7% error when combining host and device perforation with mixed precision. The binary hypervolume metric confirms that the SYprox Pareto front always covers a larger region of the objective space compared to prior approaches, with

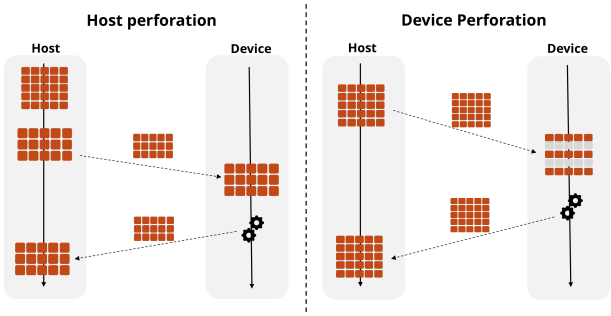

Figure 3: A SYprox example with host and device perforation.

the coverage difference indicating that SYprox consistently dominates the state of the art across all tested benchmarks. The error remains consistent across all three hardware platforms, and similar speedup and accuracy results have been obtained also on embedded GPUs. Overall, the broader context of heterogeneous computing with SYCL has been investigated in [8], discussing how all of these extensions can work together and the state and future directions of SYCL ecosystem development.

3.4 Application to Real-Time Scenarios

The integration of approximate computing with real-time constraints is a key aspect of LibreRT. At this end, thanks to SYprox support, the autotuning layer supports the parametrization of several approximation techniques (e.g., kernel perforation with different perforation schemes and skip factors and mixed precision with different data types), and allow for selecting the configuration with the best accuracy for a given real-time constraint. Although this approach is based on iterative tuning, it can easily be extended to integrate predictive tuning [10].

Validation scenarios included image processing and neural network applications for high-throughput deep- and meta-edge devices, as well as highly responsive real-time applications for micro-edge. The approximate computing techniques have been validated on standard benchmarks from image processing (Sobel, blur, Gaussian, median) and scientific computing (hotspot, lavaMD, leukocytes) domains. For image processing, benchmarks are drawn from SYCL-Bench [11] and AxBench, extended with the required SYprox annotations. SYrtos most advanced scenario is a real-world use case

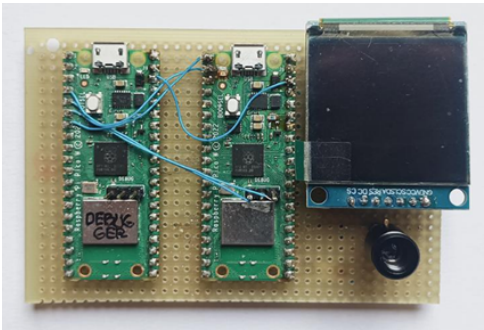

Figure 4: Thermal camera prototype.

involving a micro-controller-based system for thermal camera control, demonstrating the practicality of the approach for embedded real-time applications. Figure 4 shows the camera prototype and attached micro-controller used for real-time validation.

4 Lessons Learned

The LibreRT project has produced several insights that are relevant beyond the scope of the project itself.

SYCL is suitable for real-time and micro-controllers. Our SYCL extension for real-time showed that the SYCL standard is suitable not only for high-performance computing but also for tiny micro-controllers, and that it can seamlessly integrate real-time capabilities. The key insight is that SYCL's extensibility, combined with its single-source C++ programming model, makes it possible to add advanced real-time semantics. The SYrtos experience demonstrated that a per-queue thread pool with OS-level priority mapping provides an effective and portable mechanism for real-time scheduling.

Combining approximation techniques yields superior results. The SYprox experience highlighted the importance of combining multiple approximation techniques rather than relying on any single one. Host perforation consistently outperforms device perforation alone by reducing both computation and data transfer. Mixed precision, when composed with perforation and reconstruction, generates new Pareto-optimal solutions that dominate prior approaches. This finding motivates the design of autotuning strategies that explore the combined approximation space.

The boundary between OS kernel and application is fluid. Fluid kernels revealed that the boundary between OS kernel and application space is not as rigid as traditionally assumed in embedded systems. By allowing applications to be seamlessly compiled for either user space or kernel space, the Miosix OS can offer dramatically different performance and footprint trade-offs, with the same application code. This is especially beneficial for I/O-intensive real-time applications, where kernel-space execution can improve GPIO toggling performance by over 100×.

Portability across GPU vendors remains a practical challenge. SYCL solves most portability problems at the programming model level; however, programmers must be aware of differences in compiler maturity, backend support, and hardware capabilities across high-end GPU vendors as well as embedded GPUs. Our experimental validation on embedded GPUs, we identified specific SYCL semantics that allow performance portability and those that must be avoided to ensure efficient execution across very different architectures.

High-resolution timing need not cost performance. The 1+N timing subsystem design showed that high-resolution timekeeping (nanosecond-level) can be achieved on micro-controllers without significantly increasing context switch overhead, provided that preemption and timekeeping are decoupled using separate hardware timers. This design is generalizable to other RTOSes beyond the Miosix OS.

5 Conclusions

The LibreRT project has delivered a portable software stack addressing the full embedded computing continuum, from ultra-low-power

micro-controllers to embedded GPU-equipped meta-edge devices. At the programming model level, LibreRT extended the SYCL standard in two complementary directions: SYrtos introduces real-time capabilities through priority and deadline support for queues and kernels, while SYprox brings approximate computing to heterogeneous architectures by combining host/device perforation, signal reconstruction, and mixed precision within a single composable interface. On the operating system side, the fluid kernel architecture in Miosix bridges the traditional gap between unikernels and monolithic kernels, enabling applications to seamlessly move between kernel and user space, while a new high-resolution timing subsystem provides nanosecond-level timekeeping without penalizing context switch performance. A comprehensive portability evaluation on embedded GPUs from multiple vendors has validated the reach of SYCL across the computing continuum, and the integration of the full software stack has been demonstrated on a real-time scenario involving a thermal camera prototype.

The LibreRT project embraced the open-source approach, with a strong support for standard-based technologies ensuring that the developed technologies can be adopted and extended by the broader community.

Acknowledgments

We acknowledge financial support under the National Recovery and Resilience Plan, call for tender No. 104 published on 02/02/2022 by the Italian *Ministry of University and Research* (MUR), funded by the European Union, Next Generation EU, Mission 4, Component 1, CUP D53D23008590001 and D53D23008600006, project LibreRT.

References

[1] Aksel Alpay and Vincent Heuveline. 2020. SYCL beyond OpenCL: The architecture, current state and future direction of hipSYCL. In *Proceedings of the International Workshop on OpenCL (IWOCL)*. 1–1. doi:10.1145/3388333.3388658

[2] Ben Ashbaugh, Alexey Bader, James Brodman, Jeff Hammond, Michael Kinsner, John Pennycook, Roland Schulz, and Jason Sewall. 2020. Data parallel c++ enhancing sycl through extensions for productivity and performance. In *Proceedings of the International Workshop on OpenCL (IWOCL)*. 1–2. doi:10.1145/3388333.3388653

[3] Giorgio Audrito, Federico Terraneo, and William Fornaciari. 2023. FCPP+Miosix: Scaling Aggregate Programming to Embedded Systems. *IEEE Transactions on Parallel and Distributed Systems* 34, 3 (2023), 869–880. doi:10.1109/TPDS.2022.3232633

[4] Lorenzo Carpentieri and Biagio Cosenza. 2025. SYprox: Combining Host and Device Perforation with Mixed Precision Approximation on Heterogeneous Architectures. In *ACM International Conference on Supercomputing (ICS)*. 1–12. doi:10.1145/3721145.3725741

[5] Lorenzo Carpentieri, Marco D Antonio, Kaijie Fan, Luigi Crisci, Biagio Cosenza, Federico Ficarelli, Daniele Cesarini, Gianmarco Accordi, Davide Gadioli, Gianluca Palermo, Peter Thoman, Philip Salzmann, Philipp Gschwandtner, Markus Wippler, Filippo Marchetti, Daniele Gregori, and Andrea Rosario Beccari. 2023. Domain-Specific Energy Modeling for Drug Discovery and Magnetohydrodynamics Applications. In *International Workshop on the Environmental Sustainability of High-Performance Software (SHiPS)*. ACM, 1789–1800. doi:10.1145/3624062.3624261

[6] Lorenzo Carpentieri, Antonio De Caro, Majid Salimi Beni, Kaijie Fan, and Biagio Cosenza. 2025. Phase-based Frequency Scaling for Energy-efficient Heterogeneous Computing. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 824–836. doi:10.1109/IPDPS64566.2025.00078

[7] Lorenzo Carpentieri, Mohammad VazirPanah, and Biagio Cosenza. 2025. A Performance Analysis of Autovectorization on RVV RISC-V Boards. In *Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing (PDP)*. 129–136. doi:10.1109/PDP66500.2025.00026

[8] Biagio Cosenza, Lorenzo Carpentieri, Kaijie Fan, Marco D'Antonio, Peter Thoman, and Philip Salzmann. 2025. Toward Heterogeneous, Distributed, and Energy-Efficient Computing with SYCL. In *Scientific HPC in the pre-Exascale era (SHPC), ITADATA 2024*. https://arxiv.org/abs/2505.06022

[9] Biagio Cosenza, Giovanni De Pierro, Federico Terraneo, Daniele Cattaneo, and Giovanni Agosta. 2026. SYrtos: Extending SYCL for Real-Time Programming. In *International Workshop on OpenCL and SYCL (IWOCL)*.

[10] Biagio Cosenza, Juan J. Durillo, Stefano Ermon, and Ben Juurlink. 2017. Autotuning Stencil Computations with Structural Ordinal Regression Learning. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 287–296. doi:10.1109/IPDPS.2017.102

[11] Luigi Crisci, Lorenzo Carpentieri, Peter Thoman, Aksel Alpay, Vincent Heuveline, and Biagio Cosenza. 2024. SYCL-Bench 2020: Benchmarking SYCL 2020 on AMD, Intel, and NVIDIA GPUs. In *International Workshop on OpenCL and SYCL (IWOCL)*. Article 1, 12 pages. doi:10.1145/3648115.3648120

[12] Kaijie Fan, Marco D Antonio, Lorenzo Carpentieri, Biagio Cosenza, Federico Ficarelli, and Daniele Cesarini. 2023. SYnergy: Fine-grained Energy-Efficient Heterogeneous Computing for Scalable Energy Saving. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 69:1–69:13. doi:10.1145/3581784.3607055

[13] Alberto Leva, Federico Terraneo, Luigi Rinaldi, Alessandro Vittorio Papadopoulos, and Martina Maggio. 2016. High-Precision Low-Power Wireless Nodes' Synchronization via Decentralized Control. *IEEE Transactions on Control Systems Technology* 24, 4 (2016), 1279–1293. doi:10.1109/TCST.2015.2483559

[14] Martina Maggio, Federico Terraneo, and Alberto Leva. 2014. Task scheduling: A control-theoretical viewpoint for a general and flexible solution. 13, 4, Article 76 (March 2014), 22 pages. doi:10.1145/2560015

[15] Daniel Maier, Biagio Cosenza, and Ben Juurlink. 2018. Local memory-aware kernel perforation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization (CGO)*. doi:10.1145/3168814

[16] Konstantinos Parasyris, Giorgis Georgakoudis, Harshitha Menon, James Diffenderfer, Ignacio Laguna, Daniel Osei-Kuffuor, and Markus Schordan. 2021. HPAC: evaluating approximate computing techniques on HPC OpenMP applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. doi:10.5281/zenodo.5167980

[17] Philip Salzmann, Fabian Knorr, Peter Thoman, Philipp Gschwandtner, Biagio Cosenza, and Thomas Fahringer. 2023. An Asynchronous Dataflow-Driven Execution Model For Distributed Accelerator Computing. In *23rd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 82–93. doi:10.1109/CCGRID57682.2023.00018

[18] Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott Mahlke. 2014. Paraprox: Pattern-based approximation for data parallel applications. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems (ASPLOS)*. doi:10.1145/2541940.2541948

[19] Nicolai Stawinoga, Sohan Lal, Biagio Cosenza, Philip Salzmann, Peter Thoman, and Thomas Fahringer. 2025. A Portable Compiler-Runtime Approach for Scalability Prediction. *Future Generation Computer Systems* (2025), 108337. doi:10.1016/j.future.2025.108337

[20] The Khronos Group SYCL Working Group. 2023. SYCL Specification. https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html.

[21] Federico Terraneo and Daniele Cattaneo. 2025. Fluid Kernels: Seamlessly Conquering the Embedded Computing Continuum. *IEEE Trans. Comput.* 74, 12 (2025), 4050–4064. doi:10.1109/TC.2025.3605745

[22] Federico Terraneo and Daniele Cattaneo. 2026. Efficient Design of High-Resolution Timekeeping in Real-Time Operating Systems. In *7th Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2026)*, Hazem Ismail Ali and Harrison Kurunathan (Eds.), Vol. 140. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 4:1–4:15. doi:10.4230/OASIcs.NG-RES.2026.4

[23] Peter Thoman, Philip Salzmann, Biagio Cosenza, and Thomas Fahringer. 2019. Celerity: High-level C++ for Accelerator Clusters. In *International European Conference on Parallel and Distributed Computing (Euro-Par)*, Vol. 11725. Springer, 291–303. doi:10.1007/978-3-030-29400-7_21

[24] University of Salerno and Politecnico di Milano. 2026. LibreRT: Portable Heterogeneous Real-time Programming for the Embedded Computing Continuum. https://librert.di.unisa.it/ Accessed: 2026-04-03.

[25] Jeffrey S. Vetter, Ron Brightwell, Maya Gokhale, Pat McCormick, Rob Ross, John Shalf, Katie Antypas, David Donofrio, Travis Humble, Catherine Schuman, Brian Van Essen, Shinjae Yoo, Alex Aiken, David Bernholdt, Suren Byna, Kirk Cameron, Frank Cappello, Barbara Chapman, Andrew Chien, Mary Hall, Rebecca Hartman-Baker, Zhiling Lan, Michael Lang, John Leidel, Sherry Li, Robert Lucas, John Mellor-Crummey, Paul Peltz Jr., Thomas Peterka, Michelle Strout, and Jeremiah Wilke. 2018. *Extreme Heterogeneity 2018 - Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity*. Technical Report. U.S. Department of Energy, Office of Scientific and Technical Information. doi:10.2172/1473756