Evaluation of SYCL Suitability for High-Performance Critical Systems

Leonidas Kosmidis, Barcelona Supercomputing Center (BSC) and Universitat Politècnica de Catalunya (UPC)

Co-authors: Cristina Peralta, Matina Maria Trompouki
Outline

- Introduction and Motivation
- Background
- Software Porting
- Experimental Setup
- Evaluation
- Conclusions and Future Work
Safety Critical systems
- used in avionics, automotive and aerospace industries
- **correct** and **timely** execution is important
- any malfunction may be dangerous
- traditionally rely on very old and simple single core processors
  - Cannot provide the performance required for new advanced functionalities
Need for higher performance in Safety Critical Systems

Automotive Industry Examples:
- Advanced Driving Assistance Systems (ADAS)
  - Autonomous parking, heads-up (HUP) windshield displays and smart mirrors

Avionics
- Automatic Taxi, Take-Off and Landing (ATTOL)
Space:

- **Φ-Sat-1**
  - ESA’s technology demonstration mission, launched in 2020
  - AI accelerator, Intel Movidius Myriad 2
  - First demonstration of AI for earth observation on Hyperspectral Images

OPS-SAT-1

- ESA’s technology demonstration mission launched in 2019
- Intel/Altera Cyclone V SoC, Multicore CPU and FPGA
- Several High Performance Applications, including AI

Both Φ-Sat-2 and OPS-SAT 2 are under development
Need for higher performance in Safety Critical Systems

- Legacy hardware used for safety critical systems cannot provide the required performance
- Embedded Systems on Chip (SoC) with multicore and Graphics Processing Units (GPUs) are designed to comply with safety critical functional safety standards e.g. ISO 26262
- Very attractive candidate platforms for safety critical systems
The adoption of multicore and GPU platforms in safety critical systems require not only high performance but also ease of programmability.

Automotive functionalities are assigned a criticality level

Automotive Safety Integrity Level (ASIL)

Highest Criticality software (ASIL-D) needs to comply with certain rules:
- Restricted use of Pointers
- No dynamic memory allocation
- Static verification of program properties
- Resilience to faults
- Fault propagation

Need for high level programming models in safety critical systems

Khronos Safety Critical Systems Programming Models

- **OpenGL SC**
  - SC 1.0 – 2005
  - Fixed function graphics
  - Safety critical subset

- **OpenGL SC**
  - SC 2.0 – 2016
  - Shader programmable pipeline
  - Safety critical subset

- **OpenGL ES**
  - SC 1.0 – 2003
  - Fixed function graphics
  - Embedded systems subset

- **OpenGL ES**
  - SC 2.0 – 2007
  - Shader programmable pipeline
  - Embedded systems subset
  - Immediate mode removed

- **OpenGL**
  - SC 1.3 – 2001
  - Fixed function graphics
  - ARB assembly added
  - Immediate mode only

- **OpenGL**
  - SC 2.0 – 2004
  - Vertex and Fragment Shader
  - Programmable pipeline
  - ARB assembly removed
  - Retained mode added

- **OpenGL**
  - SC 3.0 – 2009
  - Vertex and Fragment Shader
  - Programmable pipeline
  - Immediate mode deprecated

- **OpenVX SC Extension**
  - SC Extension - 2017
  - Graph-based vision and inferencing

- **OpenVX SC**
  - OpenVX 1.3 - 2019
  - SC Extension integrated into core OpenVX specification

- **Vulkan SC**
  - SC 1.0 – 2022
  - Explicit Graphics, Compute and Display safety-critical subset

- **SYCL**
  - SC WG Created - March 2023
  - SC WG Created

- **SYCL 2020**
  - C++-based heterogeneous parallel programming

---

**Graph-based vision and inferencing**

**OpenVX SC Extension integrated into core OpenVX specification**

**Vulkan SC 1.0 - 2022**

**Explicit Graphics, Compute and Display safety-critical subset**

**SYCL SC WG Created**
Objectives

- Evaluate the applicability of SYCL for programming safety critical systems
  - Performance comparison with other parallel programming models on a candidate embedded platform
  - Assessment of programmability trade-offs
Focus on two safety critical industries, aerospace and automotive

The selected software needed to be:

- Computationally demanding
- Representative of the respective domains
- Already available in other parallel programming models
- Open Source in order to ensure reproducibility

Only software complying with this requirements were previously developed at BSC/UPC:

- GPU4S Bench/OBPMark Kernels → aerospace
- Pedestrian Detection Application → automotive
Developed during the ESA funded GPU4S (GPU for Space) project coordinated by BSC

- Partnership with Airbus Defence and Space
- Investigate the applicability of embedded GPU for space missions

Main focus

- Study the feasibility and potential benefits of using embedded GPUs for space applications
- Benchmark several embedded GPUs
- Implement a demonstrator of a space case study on an embedded GPU
Lack of benchmarks for Space

- Proprietary code, export restrictions
- Lack of GPU benchmarks for critical systems
- Definition of an open source GPU Benchmark suite: GPU4S Bench [1]
  - Building blocks from many domains identified in a space sw survey
  - ESA GPL-3 compatible license, released together with OBPMark [2]
  - Official Benchmarking suites of ESA for all types of new devices
  - Required to be used by new projects funded by ESA
  - HiPEAC Technology Transfer Award 2021

Identified building blocks and the domains they represent

<table>
<thead>
<tr>
<th>Building Block</th>
<th>Domains</th>
<th>Compression</th>
<th>Vision Based Navigation</th>
<th>Image Processing</th>
<th>Neural Network Processing</th>
<th>Signal Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Fourier Transform</td>
<td>GENEVIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ADS-B, NGDSP</td>
</tr>
<tr>
<td>Finite Impulse Response Filter</td>
<td>ADS-B, NGDSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer Wavelet Transform</td>
<td>CCSDS 122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pairwise Orthogonal Transform</td>
<td>CCSDS 122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predictor</td>
<td>CCSDS 123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrix computation</td>
<td>GENEVIS (Solver)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Image classification</td>
</tr>
<tr>
<td>Convolution Kernel</td>
<td>OpenCV</td>
<td></td>
<td>GO3S, GENEVIS</td>
<td>Image classification</td>
<td></td>
<td>ADS-B</td>
</tr>
<tr>
<td>Correlation</td>
<td>OpenCV</td>
<td></td>
<td>GO3S, GENEVIS</td>
<td>Image classification</td>
<td></td>
<td>ADS-B</td>
</tr>
<tr>
<td>Max detection</td>
<td></td>
<td></td>
<td>GO3S</td>
<td>Image classification</td>
<td></td>
<td>ADS-B</td>
</tr>
<tr>
<td>Synchronization mechanism</td>
<td>GENEVIS</td>
<td>EUCLID NIR, GO3S</td>
<td>TensorFlow</td>
<td>ADS-B, NGDSP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Allocation</td>
<td>CERES Solver, OpenCV</td>
<td>EUCLID NIR, GO3S</td>
<td>TensorFlow</td>
<td>ADS-B, NGDSP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sequential reference version for functional verification

3 parallel versions: naïve, optimised, vendor optimised libraries:

Evaluate programmability: programming effort vs performance

Implementations available in CUDA, OpenCL, HIP, OpenMP
Port benchmarks in SYCL

- Both SYCL Memory Models: Unified Shared Memory (USM) and Buffers
- Naïve and Optimised versions
- Same organization with existing GPU4S Bench ports in other parallel programming models
  - All programming models use the same program drivers and have the same overhead
  - Programming model section takes place at compile time
- Open source implementation [1]
- To be merged in the next GPU4S Bench / OBPMark Kernels release

[1] Cristina Peralta Quesada. GPU4S Bench SYCL Implementation: https://github.com/crispq95/GPU4S_Bench
Open Source application [1] developed at BSC/UPC
Developed as Multi-CPU Multi-GPU benchmark for Automotive Systems [2]
Original implementation achieved 88x times speedup over the sequential version on a server class CPU
Used 4 x86 CPUs and 2 GPUs
Ported to an embedded platform (NVIDIA Xavier) and used as a research use case and demonstrator in the UP2DATE H2020 project

Pedestrian detection on camera images

Necessary functionality required for automatic emergency breaking, mandatory since 2022 in all vehicles sold in the European Union

Implementation based on a classic vision algorithm (Viola-Jones method) instead of deep neural networks

- Explainable, easier to be used in a certified context
- Original application [1][2] written in CUDA, hand optimized to achieve high performance

Pedestrian Detection contributions in this work

- Code ported in SYCL
- Implementations in both SYCL Memory Models: Unified Shared Memory (USM) and Buffers
- 3 implementations
  - Naïve
  - In-order queues
  - Out-of-order queues
- Code available as open source [1]
  - Will be merged in the original repository

Two platforms

- A high performance platform
  - Initial target
  - Mainly for development
  - Known to support SYCL

- An embedded GPU platform
  - NVIDIA Xavier
  - Candidate platform for safety critical systems
  - Unknown support for SYCL at the beginning of the project
Experimental Setup: High Performance Platform

Hardware

- CPU: AMD Ryzen 7 1800 Eight-Core processor
- GPU: NVIDIA GeForce GTX 1080 Ti

Software

- Ubuntu 18.04.6 LTS
- CUDA, OpenCL, OpenMP
- hipSYCL v0.9.3 compiled from source
NVIDIA Xavier

- 8-core Caramel ARM v8.2 64-bit CPU
- Volta GPU with 8 Streaming Multiprocessors
- 32 GB of memory
- ISO 26262 ASIL-D Certified for use in automotive

- One of the target platforms for GPU4S project
- Main platform in H2020 UP2DATE project
Different Power Modes available

- Power budget limited to 15W due to thermal dissipation limitations in space
- Selected mode: Mode 2, 15W

Software:

- Ubuntu 18.04.6 LTS
- CUDA and OpenMP
- hipSYCL v0.9.3 compiled from source

  - No differences w.r.t. high performance platform setup
  - No issues encountered

Experimental Setup: NVIDIA Xavier
Programmability

- SYCL Buffers
  - Easier to use when development starts from scratch
  - No need for the programmer to worry about data transfers or dependencies
  - Low code complexity
    - Important for safety critical systems certification
  - Less LOC compared to USM
  - Low performance compared to USM
  - No programmability benefit for experienced programmers with low level GPU programming models
Unified Shared Memory (USM)

- Uses pointers and dynamic memory allocations
  - their use is discouraged in safety critical systems
- Major advantage of USM compared to low-level GPU programming models for safety critical systems
  - Automatically computed indices eg. in parallel_for constructs
  - Correct-by-construction
  - Reduces programming mistakes
- USM LOC similar to the lower level GPU programming models but portable across architectures
- USM provides a good trade-off between performance and programmability required for safety critical systems
GPU4S Bench Multicore CPU: SYCL vs OpenMP / High Performance Server
GPU4S Bench Multicore CPU: SYCL vs OpenMP / High Performance Server
GPU4S Bench Multicore CPU: SYCL vs OpenMP / NVIDIA Xavier
GPU4S Bench Multicore CPU: SYCL vs OpenMP / NVIDIA Xavier
GPU4S Bench Multicore GPU: SYCL vs CUDA / High Performance Server
GPU4S Bench Multicore GPU: SYCL vs CUDA / NVIDIA Xavier
GPU4S Bench Multicore GPU: SYCL vs CUDA / NVIDIA Xavier
GPU4S Bench GPU Results on Xavier
Pedestrian Detection Results

Pedestrian Detector - FPS

NVIDIA Xavier

High performance platform

15 FPS GOAL
Ported open source software from two safety critical domains to SYCL
- GPU4S Bench from the aerospace domain
- Pedestrian detection application from the automotive domain

Compared performance with implementations in other parallel programming models, OpenMP and CUDA

Evaluated on a high performance and embedded multicore and GPU platform, NVIDIA Xavier
Our work confirms that SYCL is a suitable programming model for safety critical systems
- the amount of code required for the SYCL version is less than the CUDA one
- significantly less effort is required for SYCL code development
- SYCL provides a less error prone abstraction for safety critical software development
- SYCL is portable among devices from different vendors
- SYCL provides a good trade-off of programmability and obtained performance

We can confirm that the on-going work in Khronos for the definition of the Khronos SYCL SC working group is a step towards the right direction
Future work

- Compare with Intel’s DPC++ Compatibility Tool on the same software
- Compare performance with Intel’s DPC++ SYCL implementation
- Investigate why hipSYCL’s performance of the pedestrian detection application on NVIDIA Xavier is not as competitive with CUDA as in the case of the high performance platform
- Compare performance and programmability with other high level programming models such as OpenACC
  - Preliminary comparison results available in https://upcommons.upc.edu/handle/2117/380697
- Port ESA’s Open source OBPMark Applications and ML to SYCL
- Explore the use of SYCL SC preview in the Horizon Europe METASAT project on a RISC-V based open platform GPU
  - Open source multicore CPU and GPU
Acknowledgments

This work was funded by the Ministerio de Ciencia e Innovacion - Agencia Estatal de Investigación (PID2019-107255GB-C21 and IJC-2020-045931-I MCIN/AEI/10.13039/501100011033)

the European Commission’s Horizon 2020 programme under the UP2DATE project (grant agreement 871465) and the HiPEAC Network of Excellence
Thank you!

Questions?

leonidas.kosmidis@bsc.es
Pedestrian Detection data flow graph