Accelerating Simulink/Matlab projects with SYCL

Uwe Dolinsky, Codeplay Software Limited
Company

Leaders in enabling high-performance software solutions for new AI processing systems

Enabling the toughest processors with tools and middleware based on open standards


Supported Solutions

An open, cross-industry, SYCL™ based, unified, multiarchitecture, multi-vendor programming model that delivers a common developer experience across accelerator architectures

ComputeC++

C++ platform via the SYCL™ open standard, enabling vision & machine learning e.g. TensorFlow™

ComputeAorta

The heart of Codeplay's compute technology enabling OpenCL™, SPIR-V™, HSA™ and Vulkan™

Collaborations

Enabling AI & HPC to be Open, Safe & Accessible to All

Markets

High Performance Compute (HPC)

Automotive ADAS, IoT, Cloud Compute

Smartphones & Tablets

Medical & Industrial

Technologies: Artificial Intelligence

Vision Processing

Machine Learning

Big Data Compute

And many more!
Why accelerating Simulink/Matlab with SYCL

• Many Automotive/Avionics/Defence/Embedded software projects are entirely written in Simulink/Matlab
  • Companies have large Simulink/Matlab legacy code bases
  • Engineers have mainly expertise in Matlab/Simulink using specialised commercial Matlab Simulink tool boxes
  • Targeting Safety Critical

• Taking advantage of open source tool chains and acceleration ecosystems
  • Ability to accelerate on more types of platforms via SYCL
  • Use different open libraries as backends (Eigen, Armadillo etc)
  • Ability to apply more and different verification tools to Simulink/Matlab projects

• Matlab/Simulink are widely taught in engineering and sciences

• Many research projects/codes in engineering are based on Matlab (or Matlab-like software like Octave, COMSOL, SciLab, ...)
Context 1/2: Building novel High-Performance Hybrid Batteries for Electric Vehicles

IUK-funded project (WIZer Batteries, grant no. 104427)
Collaboration led by Williams Advanced Engineering.
Codeplay’s role: Accelerating Battery Models via SYCL.

Experimental Battery Test rig at Imperial. (Image from www.imperial.ac.uk)

Embedded MPSoC platform running the BMS on the Battery. (Image from https://www.xilinx.com/products/boards-and-kits/zcu106.html)

Project partners

© 2023 Codeplay Software Ltd.
Battery Model Architecture maintained by WAE in MatLab/Simulink format

Integrating open source libraries (Eigen) and tools to read Matlab/Simulink file formats.

- Simulink/Matlab translation to SYCL/C++
- SYCL code acceleration on CPU/FPGA
What is Simulink/Matlab?

- Enables building/testing complex programs/models using a graphical environment
- Models are graphs of blocks – blocks can be simple arithmetic operations up to complex subsystems, dealing with data/storage/constants, StateFlow, Control flow, Math Operations
  - Frequently used block types: From, Goto, Inport, Outport, Reference, SubSystem, Selector, Concatenate, Terminator, Switch, RelationalOperator, MinMax, UnitDelay, Product, Sum
- Each block has inports, outports and block parameters
- Matlab/Simulink Interactions
  - Blocks can have Matlab code attached to them:
    - expressions or whole programs with multiple functions referencing external functions
  - Matlab configures workspace and can call/query Simulink model

A typical Simulink/Matlab project contains

- One or several Simulink solution files (*.slx, *mdl)
  - Simulink libraries
- Matlab source files (*.m, workspace setup, free functions, embedded/anonymous functions)
- Matlab data files
  - Matlab data files (*.dat, workspace data, simulation data, variables)
  - Data dictionaries (*.sld)
Our approach (technical and user targeted)

• Converting Simulink/Matlab solutions into C++ code (the model step) that uses an API that can be accelerated via SYCL.
• Taking advantage of open source projects to
  • Import Matlab data files (*.mat)
  • Import Simulink Solution files (*.mdl, *.slx)
  • Provide efficient Vector/Matrix/Math operations (via Eigen)
• No dependencies on MATLAB® installation
• Non-disruptive: Enabling engineers to continue to develop in Simulink/Matlab
• Option to run entire model step or individual blocks as SYCL kernels
• Only functionality required by BMS use case was implemented
Other/Related MATLAB projects

Integrating Intel® Data Analytics Acceleration Library with Matlab

CoCoSim: Open-Source verification tool for Simulink Models
• Requires MATLAB® Installation

m2cpp: Open-source tools to convert process Matlab/Simulink files
• converts Matlab files only (research project)
  https://github.com/emc2norway/m2cpp
Challenges of our Integration

• Integrating output from various open tools/libraries to process
  • Simulink solutions files
  • Data files (and dictionaries)
  • Matlab files (workspace setup files, Matlab functions (free, embedded, anonymous)

• Support different block types
  • Models consist of the main model, sub models, model references, subsystems (virtual or atomic) and are connected by ports

• Generating model step which evaluates the entire model
  • Determine execution order of blocks
  • Requires block scheduling to flatten model graph for execution

• Outputting the model and data as C++ code and integrating with backend (Eigen)

• Targeting SYCL

• Provide flexibility/configurability to enable performance tuning
Open-source projects used to help translate Simulink and Matlab files

• Simulink files read by the ConQAT project
  https://github.com/vimaier/conqat/

• Matlab data files read by MAT File Library
  https://github.com/HebiRobotics/MFL


Example Simulink model translated into C++

API header

model inports

model outports

model step function

Writing model outports

Example model (25 blocks) from https://sourceforge.net/projects/sim2c/

Model architecture rendered by open-source ConQAT library https://github.com/vimaier/conqat/
Simulink model translated into C++ - verbose block information

Example model from https://sourceforge.net/projects/sim2c/

Outputting detailed parameters of each block as C++ comments improves model introspection and debugging.
Integration of Eigen for math operations

#include "matlab_includes.h"

struct {
  double Ian1;
  double Ian2;
}

// new_mod/bbbbbbd1 [Sum, 2:1]
dcltype(auto) Val0 = cp_subtract(-2, rootImports.Ian2);
CP_LOG_PORT("new_mod/bbbbbbd1 [Sum, 2:1]", i, Val0)

// new_mod/bbbbbbd1 [Sum, 2:1]

defcltype(auto) Val0 = cp_subtract(-2, rootImports.Ian2);
CP_LOG_PORT("new_mod/bbbbbbd1 [Sum, 2:1]", i, Val0)

template <int rows, int cols, class Type>
struct cp_matrix {
  #ifdef CP_USE_EIGEN
    typedef typename std::conditional<
      (rows > AGGREGATE_THRESHOLD || cols > AGGREGATE_THRESHOLD), AggregateType,
      Eigen::Matrix<Type, rows, cols>::type
    >::type matr_type m_impl;
  #else
    // other backend
  #endif
  // declarations
};

Eigen is imported through API header
Overloads with Eigen operations selected if block argument types are types from Eigen Library
Defined inside matlab_includes.h

cp_matrix is a core data type encapsulating Eigen::Matrix
SYCL Integration

- Targeting Simulink For Each Subsystems:  

- Subsystem containing a **ForEach** block with control parameters (Dimension)
- Essentially turns a subsystem into an array of subsystems (duplicating state)
  - A subsystem is a set of blocks
  - Apply SYCL’s `parallel_for` to the array to evaluate each subsystem

- Targeting SYCL (dealing with Matlab “persistent” and “global” data inside functions – requires transforming code to hoist these variables out of the function).
SYCL Integration

Each ForEach system generates an array of subsystem structs/functors.

In SYCL mode Each subsystem in the array is evaluated in parallel.

```
struct {
  template <class T, class T2> auto operator()(T, T2) {
    // execute the model step of the ForEach Subsystem
    // return model outputs
  }
} foreach_var2[12]; //static storage

template <class Name, class T> void ModelStep(T &cgh, StepState *__state) {

dcltype(auto) fe_ret3 = process_mul(fe(cg, foreach<ForEachNum1>(__state, foreach_var2, arg1, arg2));

dcltype(auto) Val9 = cp_get_element<1>(fe_ret3);

dcltype(auto) Val10 = cp_get_element<2>(fe_ret3);
// more code
}
```

g++ file.cpp -o cpp_app
clang++ -fsycl -DUSE_SYCL file.cpp -o sycl_app
Configuring SYCL integration

• Running all For Each systems as SYCL kernels may not improve performance.

• Generated model file provides option to selectively compile functions as normal CPU functions instead.

```cpp
#include "matlab_includes.h"

// To build a kernel in SYCL mode as C++ function just define its name for example on the command line, e.g. -DForEachNum3

// Kernel names:
//   ForEachNum1
//   ForEachNum2
//   ForEachNum3
//   ForEachNum4
//   ForEachNum5

Names of the kernels generated for each For Each system

An autotuner could determine which kernels need to be SYCL kernels or CPU functions to maximise performance.
Limitations of current Simulink support

• Only a subset of Simulink Blocks are supported
• On supported blocks only a subset of block parameters are supported
• Supported Simulink/Matlab file formats
  • Simulink solution files need to be in Matlab 2020 format (or older)
    • Open source project could be extended to provide more recent file formats
    • Many Simulink projects can be converted – requires Matlab installation
  • Matlab data files need to be in format version 5 (OpenSource Octave can be used to convert data files)
• No GUI or Simulink commands supported in Matlab code
• Backend limitations: Eigen does not support all of Simulink/Matlab operations (interpolations, etc), using some unsupported Eigen modules
• Currently using synchronous execution to evaluate subsystems
• Task parallelisation opportunities not exploited yet
Future ideas

• Targeting new (updated) use-case
• Auto-tuning to decide which kernel runs where (C++, CPU, GPU, ) and with what parameters
• Asynchronous execution of SYCL kernels
• Eigen SYCL backend integration
• Generate different block schedules for performance
• Task parallelisation
• Improve Buffer creation, support USM
• Targeting SYCL to other block types
• Targeting SYCL for Safety Critical Systems
• Potentially open sourcing
• Writing paper with performance numbers on open-source use case
Wrap

• Presented open-source- based tool flow to translate Simulink/Matlab into C++ - no dependency on Matlab tools
• Enables acceleration via SYCL (For Each systems)
• Generates static model structure – deterministic model evaluation (at least in C++ without SYCL)
  • Required for automotive use case
• Successfully applied to industrial use case (Battery Models running on embedded platform)
  • Non-disruptive – enables Simulink engineers to take advantage of SYCL
• Generated C++/SYCL code provides options to:
  • evaluate the entire model step or just (For Each blocks) as SYCL kernel
  • selectively enable SYCL per individual block
Acknowledgements and disclaimer

• This work was partly supported by the Innovate UK WIZer project (grant no. 104427)

Notices & Disclaimers

Technologies may require enabled hardware, software or service activation.
No product or component can be absolutely secure.
Your costs and results may vary.
© Codeplay Software Ltd.. Codeplay, Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.
Thank You!

@codeplaysoft  info@codeplay.com  codeplay.com