Solving Maxwell’s Equations with Modern C++ and SYCL: A Case Study

Afzal Ayesha\textsuperscript{1}, Schmitt C.\textsuperscript{1}, Alhaddad S.\textsuperscript{2}, Grynko Y.\textsuperscript{2}, Teich J.\textsuperscript{1}, Förstner J.\textsuperscript{2}, Hannig F.\textsuperscript{1}

\textsuperscript{1}Hardware/Software Co-Design, Friedrich-Alexander-Universität Erlangen-Nürnberg
\textsuperscript{2}Department of Theoretical Electrical Engineering, Paderborn University

ASAP Conference, Milan, Italy, July 10, 2018
Motivation: heterogeneous computing

Running scientific computations on heterogeneous hardware devices (CPUs, GPUs, FPGAs, DSPs)?
Motivation: heterogeneous computing

Running scientific computations on heterogeneous hardware devices (CPUs, GPUs, FPGAs, DSPs)?

A standard programming model with open-source low entry cost implementation to exploits the 3Ps: Performance, Portability, Productivity
Motivation: why modern C++ (14/17/2a)?

- rapidly becoming the **language of choice for heterogeneous programming**
- **easier** (to do simple things) and **more powerful** (to do complex things)
- C++ classes to **define DSEL**, and **extensible** through keywords and attributes
- continuing to **modernize**
  - minor(/major) new version every three(/six) years
  - huge library improvements, well-defined memory model and parallelism

http://www.modernescpp.com/index.php/what-is-modern-c

```
<table>
<thead>
<tr>
<th>C++98</th>
<th>C++11</th>
<th>C++14</th>
<th>C++17</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>2011</td>
<td>2014</td>
<td>2017</td>
</tr>
</tbody>
</table>

- STL including containers and the algorithms
- Strings
- I/O Streams
- Move semantic
- Unified initialization
- auto and decltype
- Lambda functions
- Multithreading
- Regular expressions
- Smart pointers
- Hash tables
- std::array
- Reader-writer locks
- Generalized lambdas
- Fold expressions
- constexpr
- Initializers in if and switch statements
- Structured binding declarations
- Template deduction of constructors
- Guaranteed copy elision
- auto_ptr and trigraphs removed
- string_view
- Parallel algorithm of the STL
- The filesystem library
- std::any
- std::optional
- std::variant
```
Motivation: idea of SYCL?

Use modern C++ expressiveness to define a new fully standard and portable model for heterogeneous computing.
Motivation: idea of SYCL?

Use modern C++ expressiveness to define a new fully standard and portable model for heterogeneous computing
What is SYCL for?

- Open-source, royalty-free standard from Khronos Group
- Parallel system-level programming model built on the top of OpenCL (SYstemCL)
- Abstraction layer for all low-level OpenCL features
- High-level specifications expressed in modern C++ without language extension
- Single-source DESL
- Resilience to failure
- Offline kernel compilation with strong type checking
- Asynchronous task graph
- Automatic data management
SYCL implementations and ecosystem

- SYCL implementations

On-going pure C++ implementation
Started by AMD and now led by Xilinx
OpenMP for host parallelism
std::thread for task graph
Boost.Compute for OpenCL interoperability

triSYCL

Xilinx SYCL for FPGA

SYCL implementations

SYCL-gtx

ComputeCPP by Codeplay
SYCL implementations and ecosystem

- SYCL implementations

  - On-going pure C++ implementation
    - Started by AMD and now led by Xilinx
    - OpenMP for host parallelism
    - std::thread for task graph
    - Boost.Compute for OpenCL interoperability

  - triSYCL

  - Xilinx SYCL for FPGA

- SYCL ecosystem

  - parallel STL, linear algebra (Eigen, BLAS) and machine learning (TensorFlow) libraries
Agenda

further steps

• SYCL-based Maxwell’s (SBM) solver
  • Maxwell’s equation solvers
  • C-based Maxwell’s (CBM) application\(^1\) in modern C++ and SYCL
  • runtime data dependency and task graph
  • OpenCL interoperability of SYCL in SBM

• results
  • software complexity metrics
  • performance

• conclusion

• what’s next?

---

\(^1\)https://github.com/tcew/MIDG2
Maxwell’s equations solvers

Design method
Time-domain nodal discontinuous Galerkin

Domain
Unstructured meshes

Problem
Maxwell’s equations

Profiling critical kernels
Maxwell’s volume kernel, Maxwell’s surface kernel and Runge-Kutta integration kernel

Application structure

while time $< \text{maximum time}$

for each RK stage

| **Maxwell’s volume** kernel, the first half of Equations (4), (5) |
| Maxwell’s surface kernel, the second half of Equations (4), (5) |
| Runge-Kutta integration kernel, Equation (6) |

\[
e^k \frac{\partial E^k}{\partial t} = D^k \times H^k + (\mathcal{M}^k)^{-1} \mathcal{H}^k \left( \frac{\Delta E - \hat{h} \Delta E + Z^+ \hat{h} \times \Delta H}{Z} \right) \tag{4}
\]

\[
\mu^k \frac{\partial H^k}{\partial t} = -D^k \times E^k + (\mathcal{M}^k)^{-1} \mathcal{H}^k \left( \frac{\Delta H - \hat{h} \Delta H - Y^+ \hat{h} \times \Delta E}{Y} \right) \tag{5}
\]

\[
\frac{\partial}{\partial t} y(t) = g(t, y).
\]

The scheme of the LSRK is:

\[
\begin{align*}
K_1 &= y_n; \\
K_2 &= A_i K_2 + \Delta t g(t_n + c_i \Delta t, K_1) \\
K_1 &= K_1 + B_i K_2,
\end{align*}
\]

\[
y_{n+1} = K_1. \tag{6}
\]

where $s$ is the number of stages.
CBM application in modern C++ and SYCL

Modern C++ templated classes
- abstractions using standard C++ functionality
- specialized data types for vectors and matrices of any dimensionality and type
- high-level utility functions for custom data types
- interface for the unstructured meshes

Directly executable SYCL DSEL
- host fall-back and emulation for free
- debug with SYCL provided host device

Figure: Layering of SBM solver
CBM application in modern C++ and SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

buffer<real3_t, 2> b_E(range<2>{elems, 3});

queue q;

cgh.parallel_for_work_group<class fused_kernels>(
    nd_range<>(range<>(elems), range<>(grpsize)), [=](group<> grp){
    [...]
});
```

Ayesha Afzal | FAU Erlangen-Nürnberg | Solving Maxwell's Equations with Modern C++ and SYCL: A Case Study | Milan, Italy, 2018-07-10
CBM application in modern C++ and SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

buffer<real3_t, 2> b_E(range<2>{elems, 3});

queue q;

cgh.parallel_for_work_group<class fused_kernels>(
    nd_range<2>(range<2>{elems}, range<2>{grpsize}), [=](group<> grp){
        grp.parallel_for_work_item([=](nd_item<1> k){
            [...]
        });
    });
```

SYCL header contains the runtime API for SYCL objects
CBM application in modern C++ and SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

[...]
buffer<real3_t, 2> b_E(range<2>{elems, 3});
[...]
queue q;
[...]
q.submit([&] (handler &cgh) {
    auto a_E = b_E.get_access<access::mode::read>(cgh);
    [...]
    cgh.parallel_for_work_group<class fused_kernels>(
        nd_range<2>(range<2>(elems), range<1>(grpsize)), [=](group<> grp){
            [...]
        });
    });
});
```

SYCL header contains the runtime API for SYCL objects
buffers abstract location-independent data storage across host CPU and devices
CBM application in modern C++ and SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

buffer<real3_t, 2> b_E(range<2>{elems, 3});

queue q;

q.submit([&](handler &cgh) {
    auto a_E = b_E.get_access<access::mode::read>(cgh);

    cgh.parallel_for_work_group<class fused_kernels>(
        nd_range<>(range<>(elems), range<>(grpsize)), [=](group<> grp){

            grp.parallel_for_work_item([=](nd_item<1> k){

                ...
            })
        })
    }
});
```

SYCL header contains the runtime API for SYCL objects
buffers abstract location-independent data storage across host CPU and devices
queue direct computations on devices
CBM application in modern C++ and SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

buffer<real3_t, 2> b_E(range<2>{elems, 3});

queue q;

q.submit([&](handler &cgh) {
    auto a_E = b_E.get_access<access::mode::read>(cgh);
    cgh.parallel_for_work_group<class fused_kernels>(
        nd_range<>(range<>(elems), range<>(grpsize)), [=](group<> grp{
            grp.parallel_for_work_item([=](nd_item<1> k){
                // ...
            });
        });
    });
});
```

**SYCL header** contains the runtime API for SYCL objects

**buffers** abstract location-independent data storage across host CPU and devices

**queue** direct computations on devices

**command group** a collection of memory and kernel execution commands

**submit()** enqueue an asynchronous task to the queue
CBM application in modern C++ and SYCL

```c++
#include <CL/sycl.hpp>

using namespace cl::sycl;

buffer<real3_t, 2> b_E(range<2>{elems, 3});

queue q;

q.submit([&](handler &cgh) {
    auto a_E = b_E.get_access<access::mode::read>(cgh);

    cgh.parallel_for_work_group<class fused_kernels>(
        nd_range<>(range<>(elems), range<>(grpsize)), [=](group<> grp){
            grp.parallel_for_work_item([=](nd_item<1> k){
                [...}
            });
        });
    });
```

**SYCL header** contains the runtime API for SYCL objects

**buffers** abstract location-independent data storage across host CPU and devices

**queue** direct computations on devices

**command group** a collection of memory and kernel execution commands

**submit()** enqueue an asynchronous task to the queue

**accessors** express type safe access mode for buffers as read/write/read_write
CBM application in modern C++ and SYCL

```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

buffer<real3_t, 2> b_E(range<2>{elems, 3});

queue q;

q.submit([&](handler &cgh) {
  auto a_E = b_E.get_access<access::mode::read>(cgh);

  cgh.parallel_for_work_group<class fused_kernels>(
    nd_range<2>(
      range<2>(elems), range<1>(grpsize)), [=](group<> grp){
    ...
  });

  ...
});
```

SYCL header contains the runtime API for SYCL objects
buffers abstract location-independent data storage across host CPU and devices
queue direct computations on devices
command group a collection of memory and kernel execution commands
submit() enqueue an asynchronous task to the queue
accessors express type safe access mode for buffers as read/write/read_write in kernels
parallel_for automatic execution of kernels, expressed in C++, on device in different forms of parallelism, e.g., CPU-friendly hierarchical parallelism

Figure: Hierarchical data parallelism

http://archive.rtcmagazine.com/articles/view/103610
Runtime data dependency and asynchronous task graph

Automatically handling of data dependencies

- infer dependency graph directly from these accessors
- removes the need for complex event handling
- don’t need to enqueue explicit copies

Asynchronous kernels submission

- data movement optimizations
- uses as many threads and queues as necessary
- automatic overlap of kernels and communications

Figure: Efficient scheduling execution according to SBM data movement by SYCL runtime
OpenCL interoperability of SYCL in SBM

Kernel execution

```cpp
for (size_t timeStep = 0; timeStep < numSteps; ++timeStep)
for (size_t RKstage = 1; RKstage <= 5; ++RKstage)
    q.submit([&](handler &cgh) {
        cg.set_args(b_resQ.get_access(ACCESS::mode::read_write)(cgh), b_rhsQ.get_access(ACCESS::mode::read)(cgh), b_Q.get_access(ACCESS::mode::read_write)(cgh), int Ntotal, double {RKa}, double {RKb}, double {RKdt});
        cgh.parallel_for(total_size, k);
    });
```

Interaction of the abstracted C++ SYCL code with OpenCL **without overhead**

Recycle existing optimized OpenCL kernels by keeping the **high-level features of SYCL**
Results
Software complexity metrics

Reduced OpenCL device management complexity of running commands and managing the OpenCL devices at high-level with using C++ abstractions and SYCL

→ Simplified parallel programming

→ Improve programability and productivity

LoC: Lines of code; M: McCabe’s cyclomatic complexity metric;
Halstead complexity metric: N: Length; V: Volume; L: Level; E: Effort;
Performance

- **devices**: Intel Skylake i7-6700 CPU, Intel Ivy Bridge EP CPU, NVIDIA GeForce GTX 745 GPU
- **CBM solver**: ParMETIS library for mesh partitioning and MPI for communication
- **SBM solver**: No ParMETIS library for mesh partitioning or MPI for communication
Conclusion
Conclusion

SBM solver

- Opens CBM application to Khronos realm
- Relies on open-source and post-modernism
- Easier and nicer implementation by simplify plain OpenCL programming

SYCL as a foundation for building template C++ libraries
- Can use SYCL as generic heterogeneous programming model to target OpenCL, OpenMP, MPI, GPI, CUDA, OpenAMP, Vulkan, OpenGL, etc
- Specialize implementations with C++ templates for performance
- Instantiate kernels for any type within template contexts
- Sharing and reuse existing implementations

A stable, accurate and efficient implementation

Ayesha Afzal | FAU Erlangen-Nürnberg | Solving Maxwell's Equations with Modern C++ and SYCL: A Case Study
Milan, Italy, 2018-07-10
What’s next
What’s next

OpenCL interoperability mode of SYCL
to simplify usage of OpenCL built-in kernels in FPGA world

host code optimization with modern C++17 parallel STL

task-based programming model, like GPI for distributed-memory parallelization

internally handles data transfers

exposes a PGAS to the user

https://www.vectorstock.com/royalty-free-vector/a-thinking-owl-vector-1271519
Solving Maxwell’s Equations with Modern C++ and SYCL: A Case Study
Ayesha Afzal
ayesha.afzal@fau.de

Thanks for listening. Questions?

HighPerMeshes
http://highpermeshes.info
This work has been supported by the German Ministry for Education and Research (BMBF) under project grant 01IH16005
Slide for reference
Runtime data dependency and asynchronous task graph