UnoAPI: Balancing Performance, Portability, and productivity (P3) in HPC Education

George K. Thiruvathukal
Professor and Chairperson of Computer Science, Loyola University Chicago
Visiting Computer Scientist/Guest Faculty, Argonne National Laboratory Leadership Computing Facility

Konstantin Läufer
Professor of Computer Science, Loyola University Chicago

Heterogeneous Computing is (still) a Thing

- Longstanding challenge in parallel computing but resurfacing with novel architectures and accelerators
- Heterogenous CPU design remains an issue (e.g. Intel x86 vs. ARM)
- Today's challenge is more about intranode heterogeneity (CPU + one or more, possibly different, accelerators
- Not limited to exascale: IoT/Edge devices are also heterogeneous compute systems and becoming more so

The Promise of oneAPI™ and Things Like it

- A single API to program all available architectures
- Programming in a modern dialect of C++, Data-Parallel C++
- Long-time HPC folks will recognize ideas of past efforts (C* on Connection Machine, Data Parallel C)
- Built primarily on SYCL for parallelism/concurrency
- Not the only thing of its kind but likely to be in the running with strong industry support (not limited to Intel)

The UnoAPI Curriculum / Overarching Goal

Our hope is to create a curriculum that will not only teach oneAPI and its components but also result in the least amount of frustration for students and their educators, who also may need to come up to speed on the latest advancements in C++ and modern software tools that support it.

We place emphasis on well-crafted *exemplars* that embrace the best practices of systems, modern language design, and software engineering.

Meanwhile…C++ Continues to Evolve

- New foreach loop syntax to visit elements of a collection in natural order
- Improved initialization syntax, especially for unions and arrays
- Automatic variable type inference (**auto** keyword)
- Variadic templates for increased parametric code reuse
- Improved libraries for time, atomics, regex, etc.
- Threading/concurrency library (looking beyond pthreads)

Functional Programming in Modern C++

- Lambda expressions is a convenient way of defining an anonymous function object.
- Known as a closure, a lambda is passed as an argument to a function.
- A lambda can introduce new variables in its body (in **C++14**), and it can also access, or *capture*, variables from the surrounding scope.
- Lambdas eliminate the tedium of working with function pointers and extra bookkeeping for scope management
- And they improve code clarity: Yay!

See Lambda expressions in C++ | Microsoft Learn for a nice overview!

```cpp
#include <algorithm>
#include <cmath>

void abssort(float* x, unsigned n) {
    std::sort(x, x + n,
        // Lambda expression begins
        [](float a, float b) {
            return (std::abs(a) < std::abs(b));
        } // end of lambda expression
    );
}
```

Source: [Lambda expressions in C++ | Microsoft Learn]

Batteries Included is Possible in C++

- Most modern languages support thinking about dependencies.
- Python, of course, made the thinking famous via their tagline...
- ...and many other popular teaching languages (Java, Node.js, and Scala)
- By using a build system like CMake, dependencies are here to stay in C/C++.
- **Major HPC implication**: Many systems don't allow users to do sysadmin, so being able to manage dependencies is a must.

Wow, I can use CLI 11 in my C/C++ Exampes

```
1  FetchContent_Declare(
2          cli11
3          GIT_REPOSITORY https://github.com/CLIUtils/CLI11.git
4          GIT_TAG        v2.1.2
5  )
6  FetchContent_MakeAvailable(cli11)
7
8  FetchContent_Declare(
9          googletest
10         GIT_REPOSITORY https://github.com/google/googletest.git
11         GIT_TAG        release-1.11.0
12 )
13 FetchContent_MakeAvailable(googletest)
```

And I can do unit testing with Google Test!

Some Notable Third-Party Libraries (that we use)

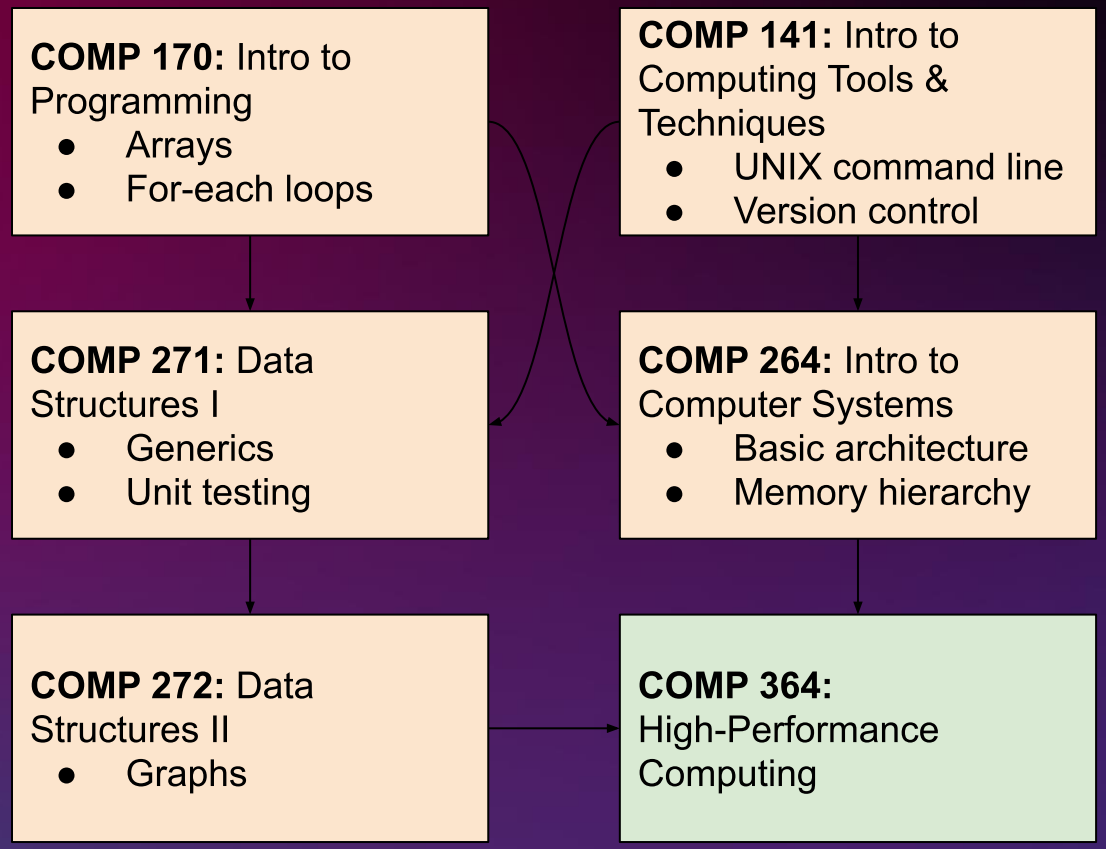
- Google Test instead of assert()/exit() in separate test programs
- CLI11 Argument Parsing instead of getopt()
- spdlog instead of printf() and cout/cerr for logging
- More elegant I/O with fmt (emerging standard) and scnlib (input)
- Our approach is to make sensible but not comprehensive choices. All batteries have options that we may use in the future
- A *curated* awesome C++ list http://github.com/fffaraz/awesome-cpp. Similar lists exist for your favorite languages, too.

Git and GitHub, Continuous Integration

- Code is available at GitHub - LoyolaChicagoCode/unoapi-dpcpp-examples.
- Book is available at https://unoapi.cs.luc.edu.
- Continuous integration checks all examples against latest oneAPI tools and runs unit tests.
- We welcome contributions and collaborations.

HPC Course and Prerequisites

- It is natural to wonder whether we can “pull this off”.
- Our curriculum at Loyola University Chicago emphasizes solid foundational preparation.
- The ACM/IEEE CS and SE curricula greatly inform our efforts.


COMP 170: Intro to Programming
● Arrays
● For-each loops
COMP 141: Intro to Computing Tools & Techniques
● UNIX command line
● Version control
COMP 271: Data Structures I
● Generics
● Unit testing
COMP 264: Intro to Computer Systems
● Basic architecture
● Memory hierarchy
COMP 272: Data Structures II
● Graphs
COMP 364: High-Performance Computing

HPC Course (COMP 364) Learning Objectives

- Learn how to analyze the scalability and efficiency of parallel algorithms and applications.
- Implement a distributed-memory parallel program using MPI and/or other messaging middleware.
- Implement a shared-memory parallel program using threads.
- Learn the hardware components and taxonomy of modern parallel computing systems, e.g. GPGPU and FPGA accelerators
- Learn to measure single-processor performance and apply optimization techniques.

How we support PPP (P3) in UnoAPI

Portability

- Build automation (PRO.cm.4, A, E)
- External dependency management (PRO.cm.3, C, E)
- Rootless package management (PRO.cm.4, A, E)
- Cross-platform standards (PRO.imp.7, K, E)

(Major.minor.section, Skill Level, E=Essential vs. D=Desirable)

PRO = Process, cm = Configuration Management

K = Know, C = Comprehend, A = Apply

Reproducibility

Automated testing (VAV.tst.11, A, E)

Continuous integration (PRO.cm.4, A, E)

VAV = Validation and Verification, PRO=Process

tst=Testing, cm=Configuration Management

A=Apply, E=Essential

Productivity

Version control (PRO.cm.1, A, E)

Modern language usage (CMP.cf.8, A, E)

Separation of concerns principle (DES.ev.1, K, E)

Software reuse (CMP.ct.2, A, E)

Software composition (PRO.pp.1, A, E)

Platform: OneAPI/SYCL data parallelism
- Device selection
- parallel_for in C++

Language: Modern C++
- const and constexpr
- auto type inference
- Lambda expressions
- Standard template library (STL)
- Performance timing

External Libraries:
- Logging
- Formatting
- Command-line parsing
- Unit testing

Technique: Automated Testing

Pedagogy: Exemplars
- Trapezoidal integration

Tool: CMake
- Build and configuration management tool
- Dependency management
- Portability

Tool: GitHub
- Version control
- Built-in CI

Tool: Rootless package management (for libraries and tools, e.g., on clusters)

Technique: Continuous Integration (CI)

A Word about UnoAPI Exemplars

- Each is a subproject of a common CMake build file (allows reuse and makes it easy for anyone to add a project)
- Has a command-line interface (CLI) to teach application-oriented thinking and do common HPC experiments (e.g. scaling)
- Use of Modern C++ throughout - no direct C usage unless it truly helps performance and improves pedagogy
- Ability to select from CPU and GPU (or others) via CLI
- Use of buffered abstractions for memory management

A Quick Tour/Demo - Time Permitting

- Code is available at GitHub - LoyolaChicagoCode/unoapi-dpcpp-examples.
- Emerging Book/Curriculum is available at https://unoapi.cs.luc.edu.
- Paper available from https://doi.org/10.6084/m9.figshare.21200464.v2.
- Continuous integration checks all examples against latest oneAPI tools and runs unit tests and builds the latest book.
- We welcome contributions and collaborations.

Evaluation and Future Plans

- Trapezoidal integration is one of several exemplars in our repo.
- Materials beta tested with small group of sophomores/juniors in SSL (research group) at Loyola and group of research students at IIT (Raicu lab)
- Will premier materials in George's next offering of COMP 339 in Spring 2022.
- Intel also includes many nice oneAPI examples. We are reworking these to incorporate our SE-focused approach.

Snippets from Exemplar – Live Demo Backup Slides

Building a Command-Line Interface (CLI) using CLI11

```
1  CLI::App app{"Trapezoidal integration"};
2
3  app.option_defaults()->always_capture_default(true);
4  app.add_option(
5    "-n,--trapezoids", number_of_trapezoids,
6    "number of trapezoids")->
7    check(CLI::PositiveNumber.description(" >= 1"));
8  app.add_option(
9    "-l,--lower,--xmin", x_min, "x min value");
10 // ...
11 app.add_option(
12   "-y,--y-format-precision", y_precision,
13   "decimal precision for y (function) values")->
14   check(CLI::PositiveNumber.description(" >= 1"));
15
16 CLI11_PARSE(app, argc, argv);
```

Building a Command-Line Interface (CLI) using CLI11

```
1     if (run_sequentially) {
2       std::vector values(size, 0.0);
3       double result{0.0};
4
5       values[0] = f(x_min);
6       for (auto i{0UL}; i < number_of_trapezoids; i++) {
7         values[i + 1] = f(x_min + i * dx);
8         result +=
9           trapezoid(values[i], values[i + 1], half_dx);
10      }
11
12      fmt::print("result = {}\n", result);
13    }
```

SYCL Device Selection – Driven by Selectable Command Line OPtion

```
1 const sycl::device_selector & device_selector{
2   run_cpuonly ?
3     static_cast<const sycl::device_selector &>(
4       sycl::cpu_selector{}) :
5     static_cast<const sycl::device_selector &>(
6       sycl::default_selector{})
7 };
```

Creating SYCL Submission Queue and Showing Selected Devices

```
1 sycl::queue q{
2   device_selector,
3   dpc_common::exception_handler,
4   sycl::property::queue::in_order()
5 };
6 spdlog::info("Device: {}", q.get_device().
7   get_info<sycl::info::device::name>());
```

SYCL's abstractions for sharing memory between devices.

```
1 sycl::buffer<double> v_buf{sycl::range<1>{size}};
2 sycl::buffer<double> r_buf{sycl::range<1>{1}};
```

Writing the parallel_for() using buffer and lambda for computing area under function f(). This computes the area but does not perform a reduce (sum).

```
1 q.submit([&](auto & h) {
2   const sycl::accessor v{v_buf, h};
3   h.parallel_for(size, [=](const auto & index) {
4     v[index] = f(x_min + index * dx);
5   }); // end inner lambda/parallel_for
6 }); // end of command group
```

The exemplar uses sycl::reduction to create a function to reduce the sum of each individual trapezoid calculation.

```
1 q.submit([&](auto & h) {
2   const sycl::accessor v{v_buf, h};
3   const auto sum_reduction{
4     sycl::reduction(r_buf, h, sycl::plus<>())};
5   h.parallel_for(
6     sycl::range<1>{number_of_trapezoids},
7     sum_reduction,
8     [=](const auto & index, auto & sum) {
9       sum.combine(
10        trapezoid(v[index], v[index + 1], half_dx));
11    } // end inner lambda
12  ); // end parallel_for
13 }); // end of outer command group
```