Embracing Modern C++ in HPC for Brain Scale Simulations

Omar Awile[1] • **Tristan Carel**[1]

[1] Blue Brain Project, Ecole Polytechnique Fédérale de Lausanne, Campus Biotech, Geneva

Blue Brain Project
Digitally reconstructing and simulating the brain

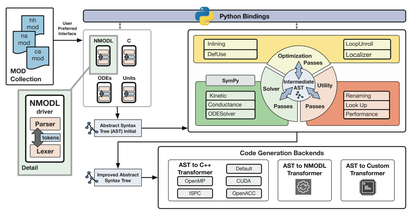
The Blue Brain Project

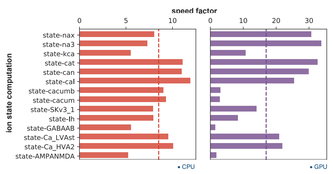
Understanding the brain is one of the largest Big Data challenges today. Neuroscience simulation is fundamental to understanding the brain as a complex multi-scale system.

The large-scale biologically-detailed simulations and digital reconstructions of rodent brain tissue built by Blue Brain offer a new approach for understanding the multilevel structure and function of the brain.

NMODL Framework

- NMODL: A 30 year old DSL used to write neuron membrane mechanisms kernels for the NEURON simulator.
- NMODL kernels are transpiled to C and compiled into simulation engine.





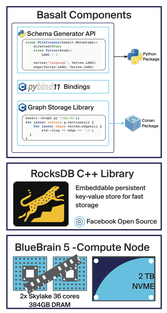
- NMODL Framework fully implements the DSL, providing:
  - High-level code analysis and optimization
  - C++, CUDA and ISPC code generation
  - A high-level Python API provides NMODL code introspection and generation for neuroscientists

https://arxiv.org/abs/1905.02241

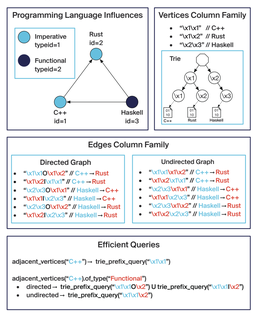
Basalt

- Modern C++ graph storage library
- Help neuroscientists manipulate data associated to brain tissue components
- Specifically tuned for NVMe devices
- Neuroscience-agnostic to KISS

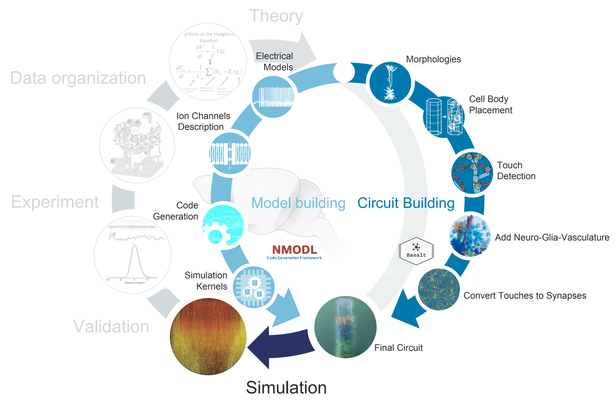
Basalt Stack



Encoding of vertices and edges for efficient lookup



The Blue Brain *in-silico* workflow



C++ idioms for HPC

Detect issues at compile-time

- Give confidence to contributors
- Detect inconsistencies with `strong type`
  - Found bugs in 20 year old C++ projects
  - No runtime overhead
- Use `override` keyword to secure usage of polymorphism.

```
template <class IntegralType, class Tag>
struct strong_id { /* ... */ };
class TetrahedronTag;
class TriangleTag;
// using triangle_id = unsigned;
// using tetrahedron_id = unsigned;
using triangle_id = strong_id<unsigned, TriangleTag>;
using tetrahedron_id = strong_id<unsigned, TetrahedronTag>;

struct StrongMesh {
    const std::vector<triangle_id>& triangle_ids() const;
    const std::vector<tetrahedron_id>& tetrahedron_ids() const;
    double volume(tetrahedron_id id) const;
    std::vector<double> volumes() const {
        //const auto& ids = triangle_ids(); // wouldn't compile
        const auto& ids = tetrahedron_ids();
        std::vector<double> volumes(ids.size());
        std::transform(ids.begin(), ids.end(), volumes.begin(),
                       [&](auto id) { return this->volume(id); });
        return volumes;
    }
};
```

Hide complexity with RAII

- Make scientific code more expressive
- Less duplication to prevent mistakes

```
using Profiler = detail::Profiler<
#if defined(ENABLE_CUDA_PROFILING)
    detail::CudaProfiler,
#endif
    detail::LikwidProfiler>;

struct scoped_profile {
    scoped_profile() { Profiler::start(); }
    ~scoped_profile() { Profiler::stop(); }
};

void do_things() {
    {
        scoped_profile prof;
        // [...] some profiled code
    }
}
```

```
namespace detail {
template <class... ProfilerImpl>
struct Profiler {
    static void start() {
        (ProfilerImpl::start(), ...);
    }
    static void stop() {
        (ProfilerImpl::stop(), ...);
    }
};

struct CudaProfiler {
    static void start();
    static void stop();
};
struct LikwidProfiler {
    static void start();
    static void stop();
};
} // namespace detail
```

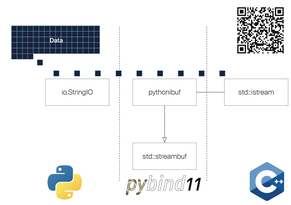
Use span when data pointers are needed

- Use span to wrap raw pointers required to use scientific libraries like Cuda or MPI
- Shared memory bugs are tedious to investigate. Using span can avoid some mistakes

```
void process_data(gsl::span<PetscScalar>& buffer);

std::vector<PetscScalar> read_and_process(Vec& a_vec) {
    std::vector<PetscScalar> res;
    PetscScalar* buf;
    PetscInt size;
    // mount vec data into buf
    VecGetLocalSize(a_vec, &size);
    VecGetArray(a_vec, &buf);

    gsl::span<PetscScalar> data(buf, size);
    res.resize(size);

    process_data(data);

    MPI_Allreduce(data.data(), res.data(), data.size(),
                  MPIU_SCALAR, MPI_MAX, MPI_COMM_WORLD);

    VecRestoreArray(a_vec, &buf);
    return res;
}
```

Stream Data when possible

- To make high-performance C++ codes accessible to scientists, we create Python bindings.
- When Python is not only used as a high-level API, data transfers between Python and C++ must be carefully considered.



Computational Neuroscience Glossary

- **Ion Channel:** Proteins arranged on cell membrane to control ion transport
- **Neuro-Glia-Vascular System:** Non-neuronal cells that support neurons
- **Cortical Column:** Group of neurons crossing the six layers of the brain (~7,900 in the mouse)
- **Cell type / em-type:** Neuron morphology and electrical behavior
- **Digital Reconstruction:** Digital model of brain tissue representing em-types and connectivity
- **Circuit Building:** Workflow for building a brain region model from individual cell morphologies
- **Model Building:** Workflow for building a model of neuron behavior from theory and measurements
- **MPI:** Message Passing Interface, a distributed-memory parallelization library for HPC applications
- **OpenMP:** a shared-memory API and library for C, C++ and Fortran applications

We're hiring!

- C++/Python Software Engineer: https://go.epfl.ch/bBw
- Scientific Developer, Numerical Methods: https://go.epfl.ch/bBx
- HPC Software Performance Engineer: https://go.epfl.ch/bBy

Lessons Learned

Use Standard File Formats
Custom config formats are not advised. Prefer JSON and YAML instead modern C++ libraries are available

Define coding conventions Tools
- clang-format, clang-tidy
- cmake-format
- pre-commit

Compiler flags: GCC, Clang, AppleClang Conventions for
- Coding style
- Naming conventions
- Documentation
- Good practices

Leverage the C++ community
- Use from the beginning community components for basic tasks e.g. fmt, spdlog, cli11, Eigen

Reduce the knowledge gap for the scientist
Provide to the scientific programmer accessible interfaces to high-performance codes through:
- Widely used DSLs
- Python bindings
- Consistent and interoperable APIs

Provide to the scientific user easy access through:
- Spack packages and lmod/tclmod
- Docker images
- pypi wheels including the pre-compiled C++ backends