SpECTRE: A Next-Generation Relativistic Astrophysics Code

Nils Deppe
Simulating eXtreme Spacetimes Collaboration

Charm++ Workshop

April 18, 2018

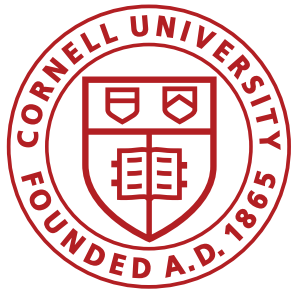


Table of Contents

SpECTRE Goals

Physics:

- Multi-scale, multi-physics in astrophysics
- Binary black holes, neutron stars
- Core-collapse supernovae with micro-physics
- Multi-disciplinary

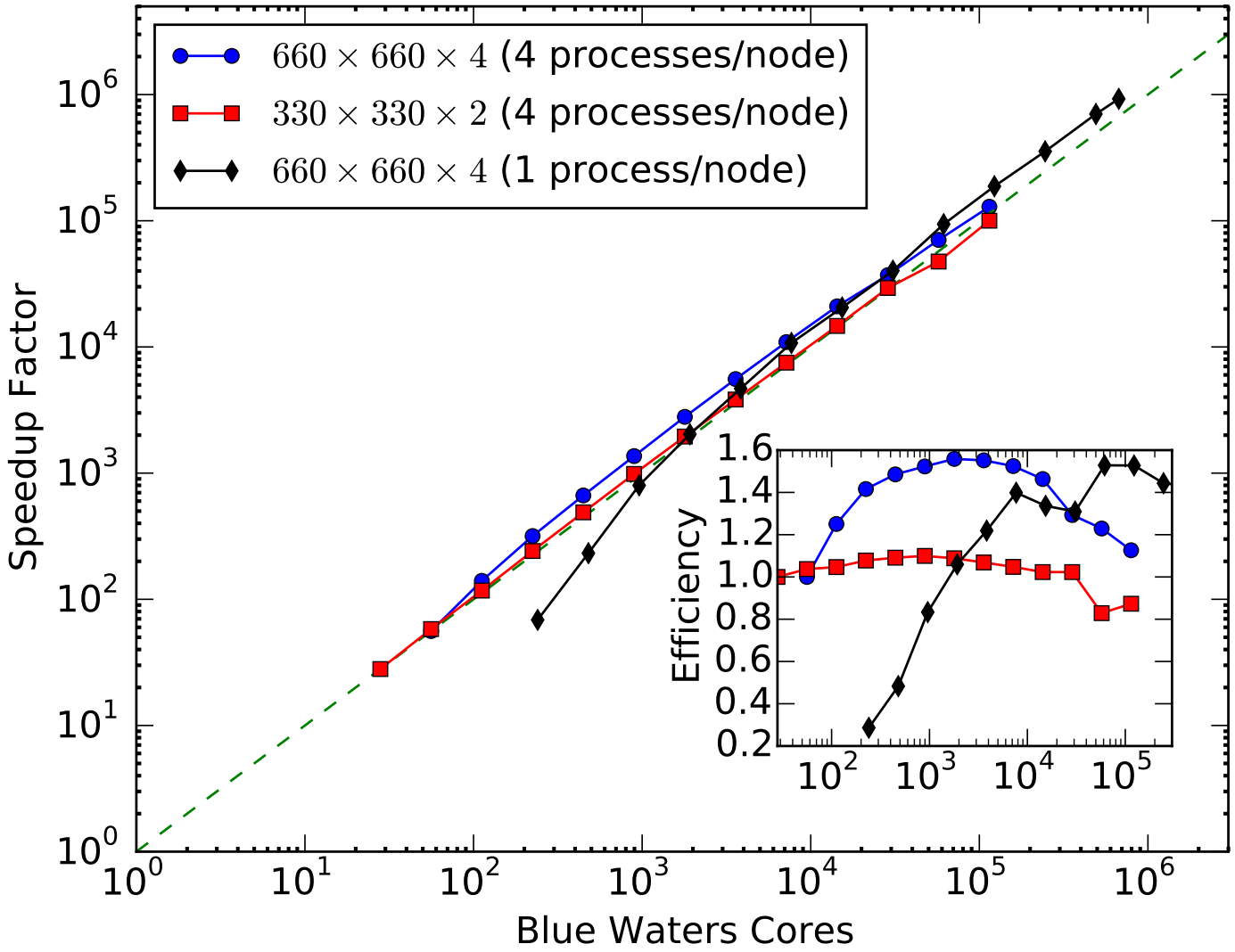
HPC:

- Open-source
- Efficient
- Exascale

SpECTRE: Previous Generation

- Charm++ for parallelization
- Solve PDEs using discontinuous Galerkin + method of lines
- Uniform, Cartesian grid
- Scalar waves, Newtonian + relativistic Euler, relativistic MHD
- Successful study of task-based parallelism + dG
- Closed-source toy code
- Paper: Kidder *et. al*, arXiv: 1609.00098, J Comp Phys

Scaling Results



Kidder *et. al*, arXiv: 1609.00098

Desired Features for Next Generation

- Abstract method for communication
- Code modularity
- Type-safe data retrieval
- Almost no run-time errors

The Solution

Container requirements:

- Container to hold disparate types
- Know types at compile-time
- Efficient lookup, preferably at compile-time

Solutions:

- `unordered_map<string, boost::any>`
- `std::tuple<...>`

Serious drawbacks

Table of Contents

TMP Example

Problem: Iterate sequences and associative container
Choose right function:

```
template <class T,
  typename enable_if_t<is_associative<T>::value>*>
double function(const T& v) {...}

template <class T,
  typename enable_if_t<is_sequence<T>::value>*>
double function(const T& v) {...}
```

Why You Should Care

Pros:

- Generic code
- Computations at compile time
- Catch errors early
- Domain specific languages
- Growing field

Cons:

- "Difficult" to learn (being improved)
- Difficult to debug (templight on GitHub)
- Difficult to optimize (templight on GitHub)

Tags

- "string" in map
- Can be template

```
struct MyTag {
  using type =
    unordered_map<std::string, double>;
};
```

TMP Basic Concepts

Sequence:

```
template <typename...>
struct typelist {};

using my_list = typelist<Tag0, Tag1, Tag2, Tag3>;
```

Retrieve Tag location in tuple:

```
template <typename Tag>
using tag_index = index_of<my_list, Tag>;
```

Tagged Tuple

Answer: Compile-time "map"

```
template <typename... Tags>
struct TaggedTuple :
  std::tuple<typename Tags::type...>
{
  using tag_ls = typelist<Tags...>;

  template <typename Tag>
  typename Tag::type& get() {
    return std::get<index_of<tag_ls,
                             Tag>::value>(*this);
  }
};
```

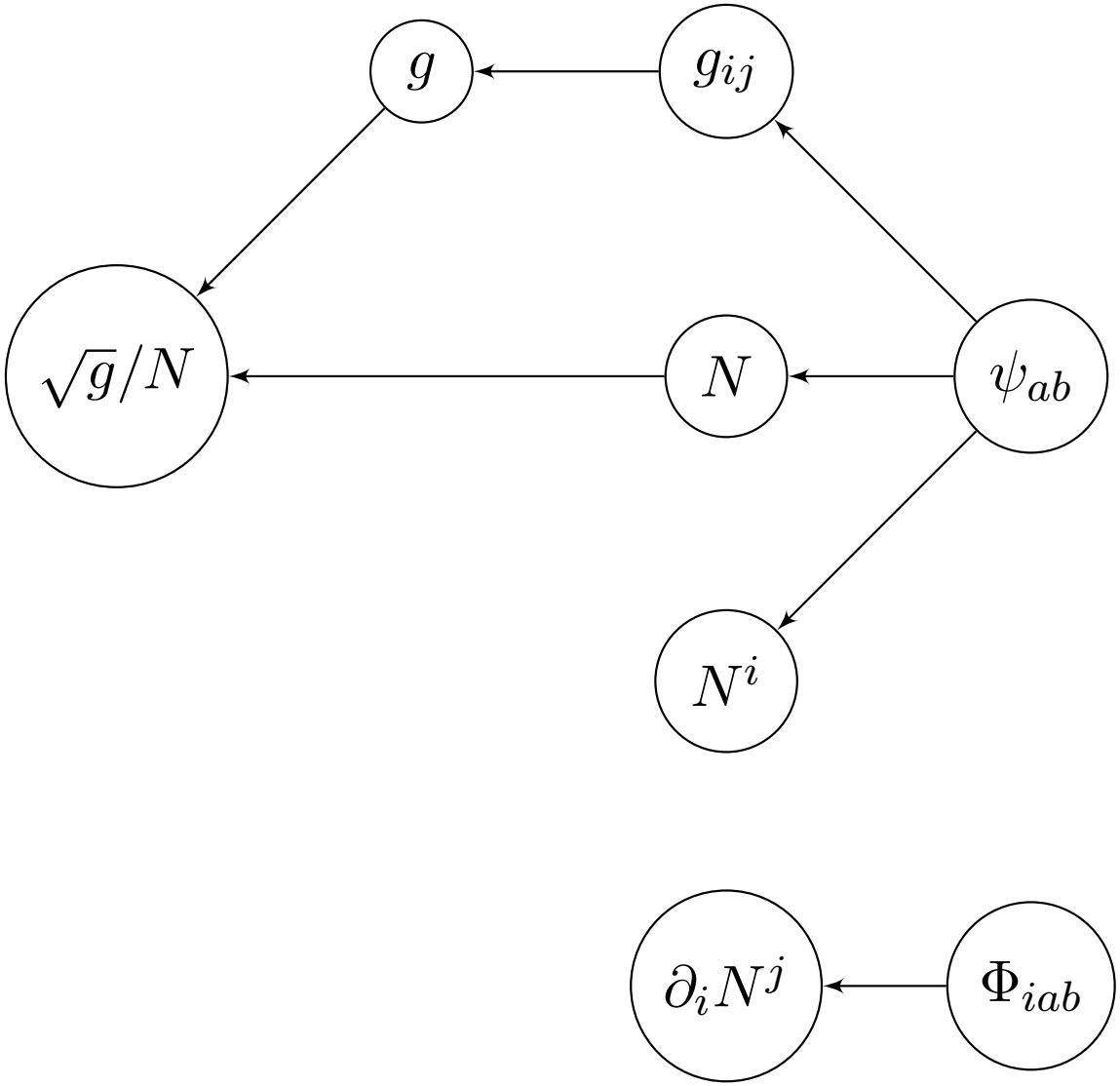
Table of Contents

DataBox

Requirements:

- Lazy evaluation
- Dependency analysis in evaluation
- Lazily recompute on dependency change

DataBox Dependency Example



DataBox Solution

- std::shared_future for lazy evaluation
- Compile-time digraph for dependency analysis
- Add-remove creates new DataBox, lightweight pointer copy

Tentacle Setup

- Multi-phase initialization
- Complex communication during initialization
- Start simulation after initialization

Use type traits and TMP
Tentacle Tag with initialization functions
Startup function in Tentacle

```
using TentaclesList = typelist<Observers,
                               Elements,
                               Main>;
```

Tentacle Usage

Retrieve observer on this node:

```
auto* local_observer =
    const_global_cache.template get<Observers>()
        .ckLocalBranch();
```

Use `auto` keyword

No `dynamic_cast`

SpECTRE: Current Generation

- Charm++ for parallelization
- Non-uniform grids
- Very modular
- TMP for better errors, easier development and maintenance
- Open sourcing in-progress on GitHub (sxs-collaboration/spectre)

Charm++ Wish List

- Implement our C++11/14 patches into Charm++
- Better TMP support in Charm++
- Replace interface files with TMP