META
A Toolkit for Template Metaprogramming

Christopher Taylor, Ryan Kabrick,
Chase Keller
John Leidel, David Donofrio
Tactical Computing Labs

Tactical Computing Labs

- Research & Development Firm
- Specialties
  - Scientific computing & HPC/Supercomputing solutions and applications
  - Numerical algorithms/libraries, Compilers, Runtime systems
  - Custom hardware design and simulation: RISC-V

Overview

Tactical Computing Labs

- Background
- Challenge
- META
- Output Kernels

Background

Tactical Computing Labs

- HPC acquisitions are expensive and have second order costs
  - Application portability is a critical issue
    - Application codes, runtime systems, and support libraries need to be migrated to new platforms
  - Custom compiler work is expensive and time consuming
    - Engaging with vendors requires demonstrating overlap with general purpose users to reduce costs

Background

- HPC C++ Developers supporting scientists often implement Domain Specific Libraries
  - Libraries are presented to scientists using C++ Domain Specific Languages
  - The goal is to provide high performance, productivity, and portability
  - Solutions tend to emphasize streamlining the *writing process*

Background

- HPC C++ Domain Specific Libraries and Languages
  - Often exercise *Template Meta Programming* (TMP) implementation techniques
  - TMP provides unique levels of customization & abstraction
  - TMP is great in high performance settings (ex: EVE simd library)
  - TMP difficult to implement and debug; requiring advanced, specialized, skill set
  - Detecting performance portability issues migrating to new HPC systems is difficult

Background

- Two examples of domain specific libraries and languages
  - Lawerence Livermore National Lab’s RAJA
  - Sandia National Lab’s Kokkos
- Both offer users a “sandbox” of functionality, for local task and data parallelism
- Users can select different runtime system backends
  - pthreads, OpenMP, TBB, Cilk, Qthreads, HPX, NVIDIA’s CUDA, or AMD HIP

Challenge

- C++ domain specific library/languages provide no additional context to the compiler
  - The compiler cannot understand the nuance of abstractions
  - Template expansion errors are difficult to read; time consuming to understand
  - Potential to yield “silent” performance loss and impediments
  - Direct impact on performance portability; there is a disconnect

Challenge

- What would a solution look like?
  - Custom static analysis of the application’s use of the domain specific library or language
  - Impose rules over use of the domain specific library or language
  - Analyzer that takes into account descriptions of hardware

META

- Clang plug-in that implements a variety of rules over C++ domain specific languages and libraries
  - Extensible (ex: ISO C++ STL task and data parallel support)
  - Currently supports RAJA and Kokkos
  - Creates parallel data flow and parallel control flow graphs
  - Performs analysis using customized compiler passes

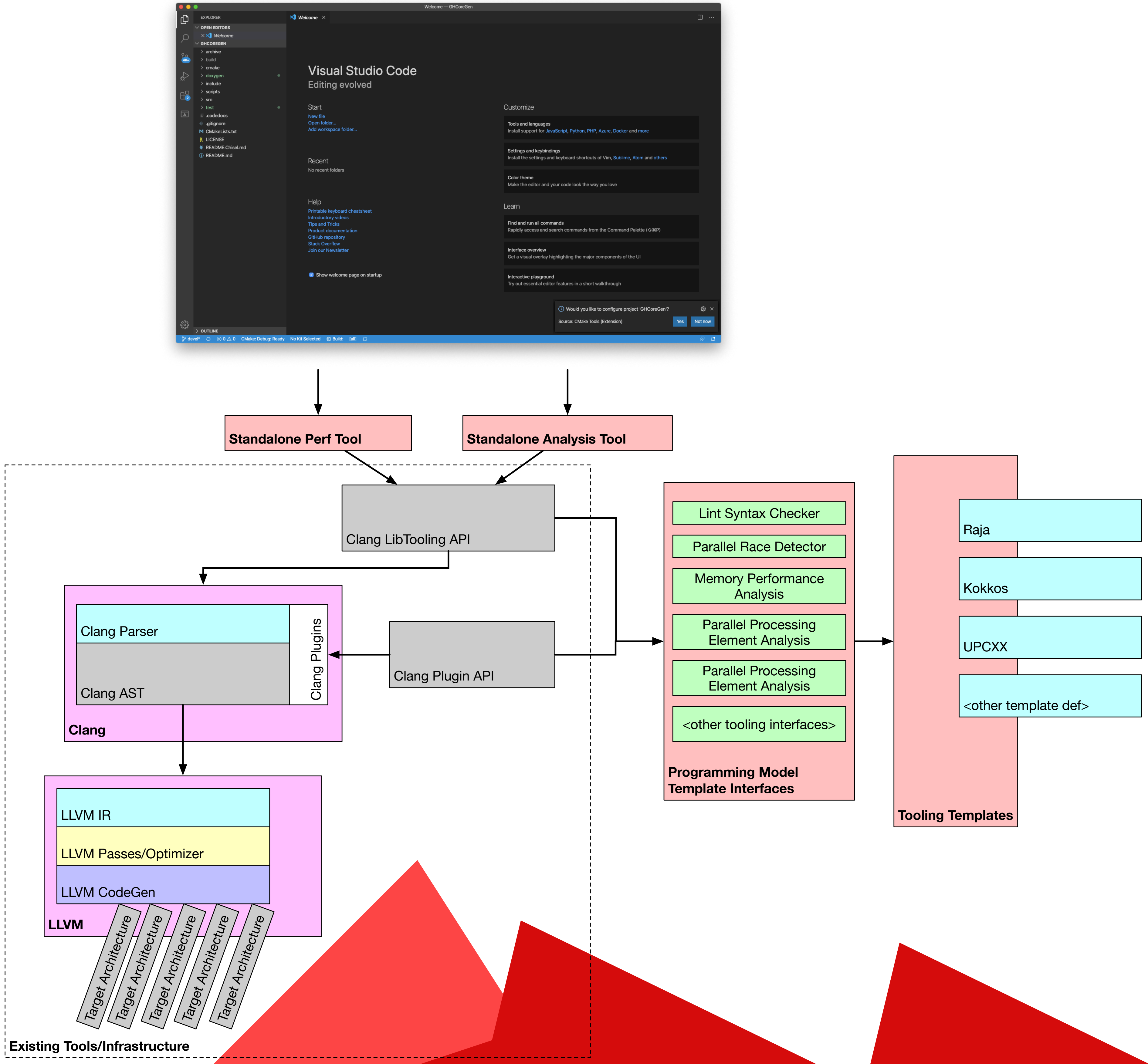
META

- Makes heavy use of Clang ASTMatchers
  - Tree regex system to find Clang AST information
  - AST matches *stream* into the plugin system in no discernible order

META

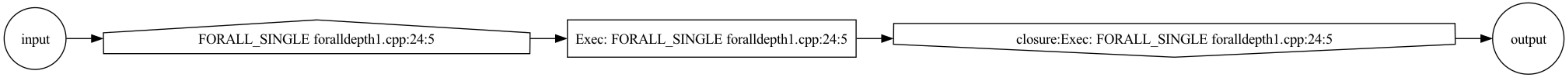
- Support hardware configuration profiles
  - Allows META to consider the hardware and system architecture
  - Bridges the gap between TMP abstractions and the underlying machine

META



Visual Studio Code
Editing evolved
Standalone Perf Tool
Standalone Analysis Tool
Clang LibTooling API
Clang Parser
Clang AST
Clang Plugins
Clang
Clang Plugin API
LLVM IR
LLVM Passes/Optimizer
LLVM CodeGen
LLVM
Target Architecture
Target Architecture
Target Architecture
Target Architecture
Target Architecture
Existing Tools/Infrastructure
Lint Syntax Checker
Parallel Race Detector
Memory Performance Analysis
Parallel Processing Element Analysis
Parallel Processing Element Analysis
<other tooling interfaces>
Programming Model Template Interfaces
Raja
Kokkos
UPCXX
<other template def>
Tooling Templates

META - Parallel Control Flow Graph



- ClangAST matchers are used to construct the PCFG
  - Mandatory *input* and *output* nodes
  - Parallel or Serial nodes are added: Entry, Scope, Exit
  - Syntactic data is stored in the nodes for analysis passes

Analysis Passes

- Nested parallelism detection
- Parallelism width analysis
- Serialization detection
- Race condition detection

Analysis Passes

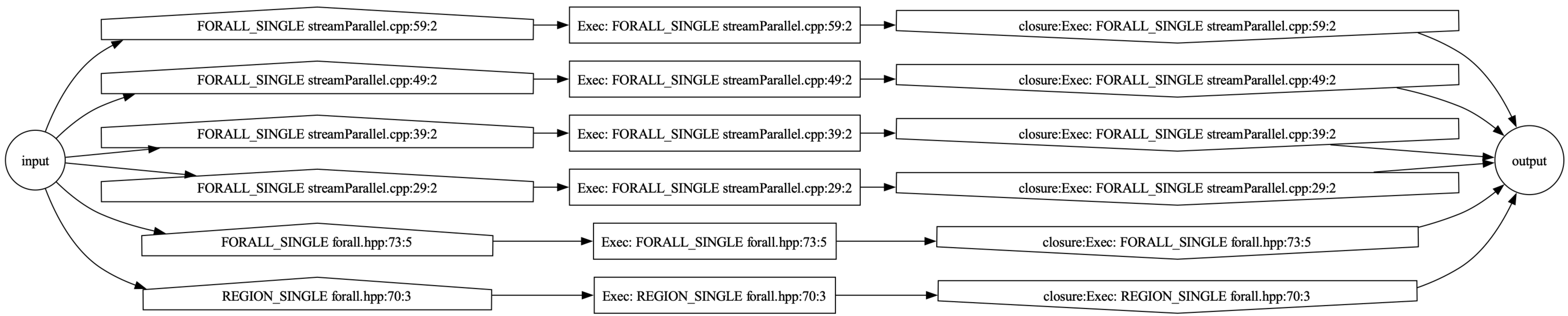
- Nested Parallelism Detection
  - Detect oversubscribed hardware (parallel loop hierarchies)
- Parallel Width Analysis
  - Inspects sequential length, verifies loops can be mapped in parallel over a CPU or Accelerator (GPU, etc) using hardware configuration profiles
- Serialization Detection
  - Depth first search of PCFG to find sets of serial dispatch nodes

Analysis Passes

- Race Condition Detection
  - Creates a Program Execution Graph (PG) segmenting code blocks by parallel scopes or asynchronous dispatch
  - Create symbol table mapping variable declarations to input scope or parallel scopes
  - A cograph is composed from the PG using the symbol table
  - The cograph indicates where race conditions occur
  - Balsundaram & Kennedy, “Compile-time detection of race conditions in a parallel program”

Output Kernels

STREAM RAJA



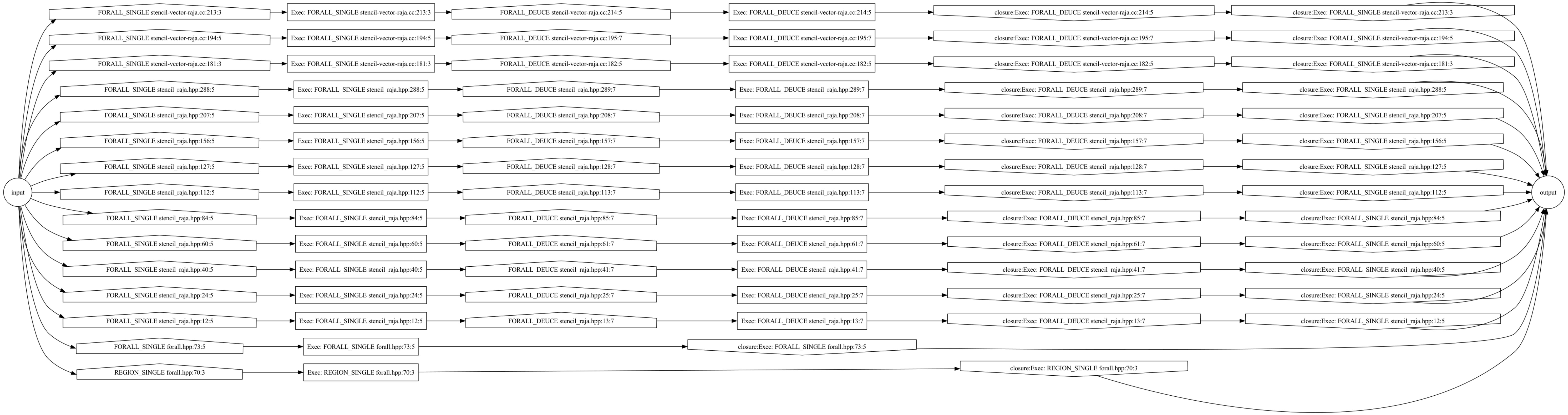

```
// copy
RAJA::forall<RAJA::omp_parallel_for_exec>(RAJA::RangeSegment(0, N), [&](int i){
    c[i] = a[i];
});

//scale
RAJA::forall<RAJA::omp_parallel_for_exec>(RAJA::RangeSegment(0, N), [&](int i){
    c[i] = 3.0f * a[i];
});
```

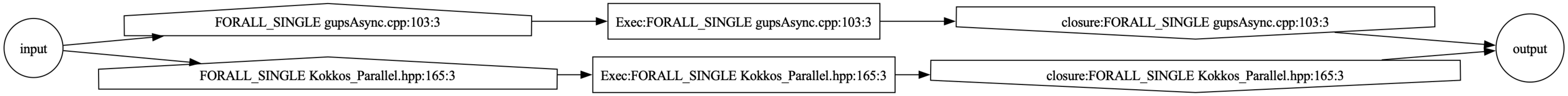
```
//add
RAJA::forall<RAJA::omp_parallel_for_exec>(RAJA::RangeSegment(0, N), [&](int i){
    c[i] = a[i] + b[i];
});

//triad
RAJA::forall<RAJA::omp_parallel_for_exec>(RAJA::RangeSegment(0, N), [&](int i){
    d[i] = a[i] + 3.0f * b[i];
});
```

PAR RES STENCIL RAJA



GUPS Kokkos Async




```
template<class Scheduler>
struct Gups{
        …

        template <class MemberType>
         KOKKOS_INLINE_FUNCTION
         void operator()(MemberType &member, int& result) const {

                Kokkos::parallel_for("gups",z,KOKKOS_LAMBDA(const int r){
                        TView[randval[r]&indexMask] ^= randval[r];
                });

         }
};

Kokkos::host_spawn(…, Gups<scheduler_type>{…});
Kokkos::wait(root);
```

GUPS Kokkos Async

- Requires “slack” Clang ASTMatchers
- Requires construction of a “reduced form AST”
- Graph construction requires “interpreter style” analysis

```
template<class Scheduler>
struct Gups{
	…

	template <class MemberType>
	 KOKKOS_INLINE_FUNCTION
	 void operator()(MemberType &member, int& result) const {

		Kokkos::parallel_for("gups",z,KOKKOS_LAMBDA(const int r){
			TView[randval[r]&indexMask] ^= randval[r];
		});

	 }
};

Kokkos::host_spawn(…, Gups<scheduler_type>{…});
Kokkos::wait(root);
```

Function & Recursion Challenge

```
void fun(…) {
    RAJA::forall<RAJA::omp_parallel_for_exec>(RAJA::RangeSegment(0, N), [&](int i){
        fun(…)
    });
}

int main(int argc, char ** arg) {
    RAJA::forall<RAJA::omp_parallel_for_exec>(RAJA::RangeSegment(0, N), [&](int i){
        fun(…)
    });
}
```

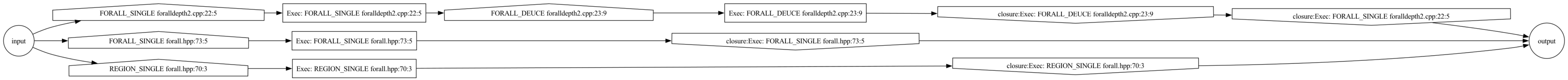
Function & Recursion Challenge

- Recursion and Clang ASTMatching complicate processing
- All information needs to be collected before graphs can be constructed
- Requires a reduced AST representation of the application
- Additional passes to process

```
void fun(…) {
  RAJA::forall<RAJA::omp_parallel_for_exec>(RAJA::RangeSegment(0, N),[&](int i){
     fun(…)
  });
}

RAJA::forall<RAJA::omp_parallel_for_exec>(RAJA::RangeSegment(0, N), [&](int i){
   fun(…)
});
```

Error Discovery




```
RAJA::forall<RAJA::omp_parallel_exec>(RAJA::RangeSegment(0, 10), [&](int j){
    RAJA::forall<RAJA::seq_exec>(RAJA::RangeSegment(0, 10), [&](int i){
        c[i] = a[i] + b[i];
    });
});
```

```
Serialization:Found serialized parallel node at : ../data/RAJA/forall/foralldepth2.cpp:23:9
Serialization:Found serialized parallel node at : /Users/ctaylor/git/install/include/RAJA/policy/openmp/forall.hpp:73:5
NestedPar:Found inverted parallel regions, consider region fusion or invert the structure; Parent region at : ../data/RAJA/forall/foralldepth2.cpp:22:5; Child region at : ../data/RAJA/forall/foralldepth2.cpp:22:5
ParWidth:Found parallel host node with insufficient parallelism (10) at : ../data/RAJA/forall/foralldepth2.cpp:22:5
```

Thank You!

TCL