OpenACC support in Flang with a MLIR dialect

Valentin Clement, Jefferey S. Vetter

2020 Virtual LLVM Developers' Meeting
October 7th, 2020

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

http://ft.ornl.gov

Acknowledgement

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

What is Flacc?

- Goal
  - OpenACC support for Flang and LLVM
- Design
  - Lower AST to a mix of FIR and OpenACC MLIR Dialects
- Availability
  - Upstream as it is developed
- Funding
  - US Exascale Computing Project (ECP)
- Contact
  - Valentin Clement (clementv@ornl.gov)

OpenACC

- Launch in 2010 as a portable directive-based programming model for C, C++, Fortran for heterogenous accelerators
- Best known for NVIDIA GPU; implementations have targeted AMD GCN, multicore CPU, Intel Xeon Phi, FPGA
- Compared to OpenMP
  - Descriptive vs. Prescriptive
  - Many features ported to OpenMP
  - Specification less complex
- OpenACC 3.0 released Nov. 2019

Roadmap

Late 2019 and 2020

- OpenACC parser and semantic checks
- OpenACC 3.0 MLIR dialect
- Flang OpenACC AST lowering to MLIR
- Generalization of common OpenMP and OpenACC infrastructure in Flang and LLVM.

Late 2020 and later

- OpenACC MLIR dialect lowering
- OpenACC runtime
- Optimization

Upstream Contributions

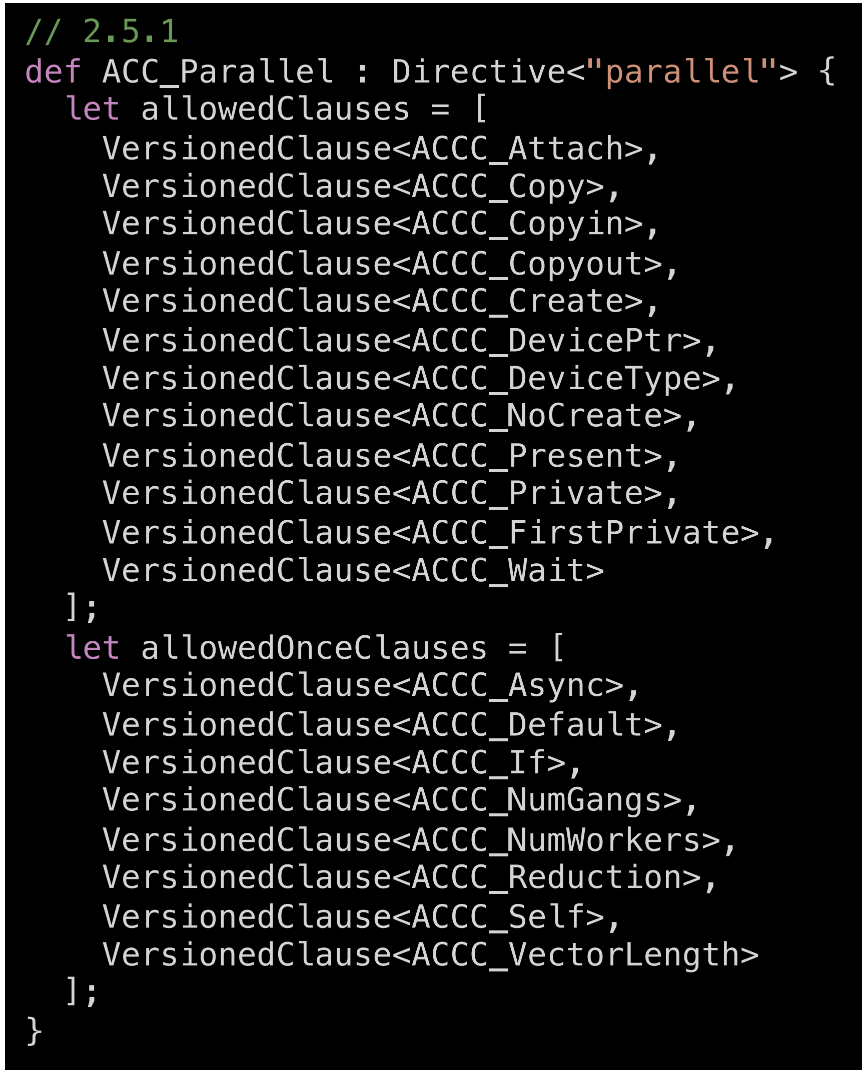
- OpenACC 3.0 parser for Flang
- OpenACC 3.0 semantic checks for Flang
- TableGen backend for Directive based language
- OpenACC MLIR dialect (WIP)

Upstream contributions – OpenACC 3.0 parser + sema

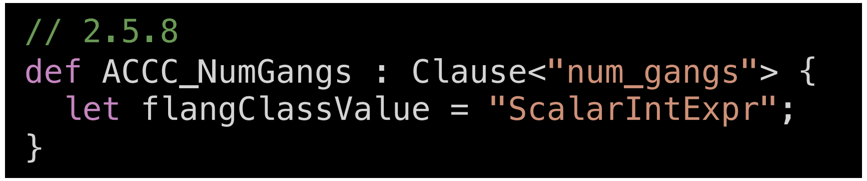
Full OpenACC 3.0 parser with un-parsing capability

- AST nodes
  - `flang/include/flang/Parser/parse-tree.h`
- Parser
  - `flang/lib/Parser/openacc-parsers.cpp`
- Semantic
  - `flang/lib/Semantics/check-acc-structure.h`
  - `flang/lib/Semantics/check-acc-structure.cpp`
  - `flang/lib/Semantics/canonicalize-acc.h`
  - `flang/lib/Semantics/canonicalize-acc.cpp`

Upstream contributions - TableGen



```
// 2.5.1
def ACC_Parallel : Directive<"parallel"> {
  let allowedClauses = [
    VersionedClause<ACCC_Attach>,
    VersionedClause<ACCC_Copy>,
    VersionedClause<ACCC_Copyin>,
    VersionedClause<ACCC_Copyout>,
    VersionedClause<ACCC_Create>,
    VersionedClause<ACCC_DevicePtr>,
    VersionedClause<ACCC_DeviceType>,
    VersionedClause<ACCC_NoCreate>,
    VersionedClause<ACCC_Present>,
    VersionedClause<ACCC_Private>,
    VersionedClause<ACCC_FirstPrivate>,
    VersionedClause<ACCC_Wait>
  ];
  let allowedOnceClauses = [
    VersionedClause<ACCC_Async>,
    VersionedClause<ACCC_Default>,
    VersionedClause<ACCC_If>,
    VersionedClause<ACCC_NumGangs>,
    VersionedClause<ACCC_NumWorkers>,
    VersionedClause<ACCC_Reduction>,
    VersionedClause<ACCC_Self>,
    VersionedClause<ACCC_VectorLength>
  ];
}
```



```
// 2.5.8
def ACCC_NumGangs : Clause<"num_gangs"> {
  let flangClassValue = "ScalarIntExpr";
}
```

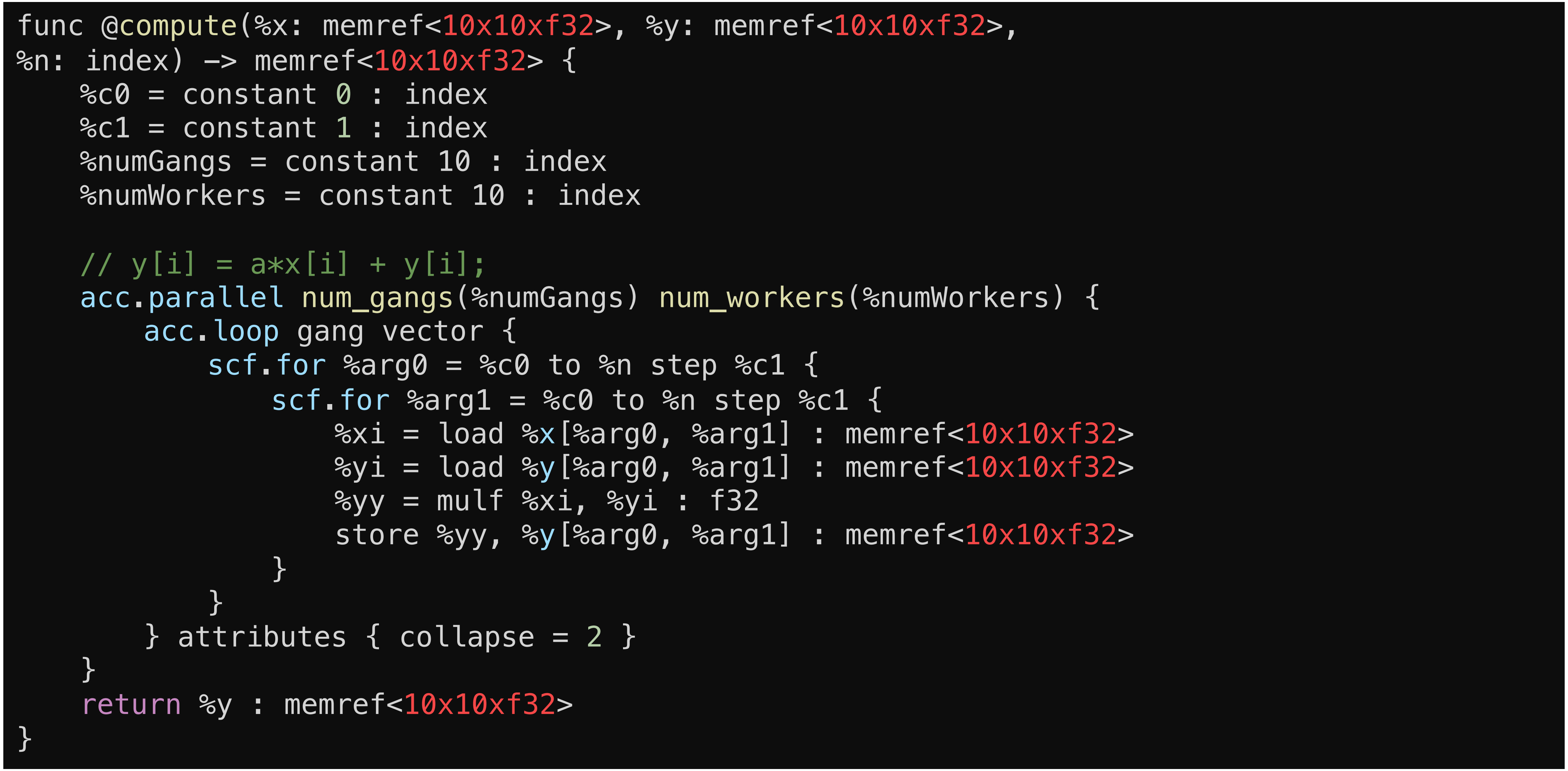
TableGen backend

- llvm/include/llvm/TableGen/DirectiveEmitter.h
- llvm/utils/TableGen/DirectiveEmitter.cpp

TableGen files for the base, OpenACC, OpenMP

- llvm/include/llvm/Frontend/Directive/DirectiveBase.td
- llvm/include/llvm/Frontend/OpenACC/ACC.td
- llvm/include/llvm/Frontend/OpenMP/OMP.td

Upstream contributions – OpenACC MLIR dialect



```
func @compute(%x: memref<10x10xf32>, %y: memref<10x10xf32>,
%n: index) -> memref<10x10xf32> {
    %c0 = constant 0 : index
    %c1 = constant 1 : index
    %numGangs = constant 10 : index
    %numWorkers = constant 10 : index

    // y[i] = a*x[i] + y[i];
    acc.parallel num_gangs(%numGangs) num_workers(%numWorkers) {
        acc.loop gang vector {
            scf.for %arg0 = %c0 to %n step %c1 {
                scf.for %arg1 = %c0 to %n step %c1 {
                    %xi = load %x[%arg0, %arg1] : memref<10x10xf32>
                    %yi = load %y[%arg0, %arg1] : memref<10x10xf32>
                    %yy = mulf %xi, %yi : f32
                    store %yy, %y[%arg0, %arg1] : memref<10x10xf32>
                }
            }
        } attributes { collapse = 2 }
    }
    return %y : memref<10x10xf32>
}
```

OpenACC Support Takeaways

- Overview
  - Goal: OpenACC support for Flang and LLVM
  - Design: Translate to an OpenACC dialect
  - Availability: Upstream LLVM
  - Contact: Valentin Clement (clementv@ornl.gov)
- Join Us
  - Oak Ridge National Laboratory
  - Hiring interns, postdocs, research and technical staff
  - External collaborators welcome