How to Make Hardware with Maths: An Introduction to CIRCT's Scheduling Infrastructure

Julian Oppermann, TU Darmstadt
Mike Urbach, SiFive
John Demme, Microsoft

European LLVM Developers' Meeting — London, UK — May 11, 2022
CIRCT

= Circuit IR Compilers and Tools

- MLIR-based compiler infrastructure for hardware design and verification
- LLVM incubator project
- More info @ US DevMtg 2021
  - “CIRCT: Lifting hardware development out of the 20th century”
  - “Charting CIRCT: the present and near future landscape”
High-level Synthesis (seen from orbit)

```c
func @dot(%X: memref<64x132>,
      %Y: memref<64x132>) -> i32 {
    %c0_i32 = arith.constant 0 : i32
    %0 = affine.for %i = 0 to 64
        iter_args(%sum = %c0_i32) -> (i32) {
        %ldX = affine.load %X[%i] : memref<64x132>
        %ldY = affine.load %Y[%i] : memref<64x132>
        %mul = arith.muli %ldX, %ldY : i32
        %add = arith.addi %sum, %mul : i32
        affine.yield %add : i32
    }
    return %0 : i32
}
```

Software: Program executes sequentially. “Untimed”.

Hardware: Everything is parallel and always running.

Scheduling (here):
- ... when to start each hardware module
- ... and how often we can load new data into the pipeline
Why does CIRCT need scheduling infra?

• Predominant abstraction is the Register-Transfer-Level (RTL)
  • Modules, wires, registers, clocks, …
  • Hard to transform – progression of time baked into the structural design

• Lifting the abstraction level is crucial for 21st century tools
  • CIRCT is a great playground for that

• Higher-level IRs are often untimed
  • Dataflow graphs, affine loops, systolic arrays, …
  • Easy to transform and suitable for design-space exploration!
  • At some point, lower to latency-insensitive hardware (handshakes), or schedule at compile time (and synthesise controller)
Goals and audience

• Provide infrastructure that is as flexible as CIRCT itself
  • Problem model should be tailored to source IR and target architecture

• Audience
  • People that find scheduling boring: Grow a library of ready-to-use problem definitions and suitable scheduling algorithms
  • People that find scheduling exciting: Foster research into algorithms by providing consistent API to hook into practically-relevant hardware-design flows
Cyclic scheduling problem

- Components
  - Operations
  - Dependences
  - Operator types

- Properties
  - Input
  - Solution

- Constraints
  - Input
  - Solution

All operations are linked to an operator type with a latency property

\[ \forall \text{ dependences } d \text{ from } i \text{ to } j: \]
\[ i.\text{startTime} + i.\text{opr.latency} \leq j.\text{startTime} + d.\text{distance} \times II \]
Extensible problem model

• Different flows require different problem variants
  • Properties + Constraints = Reliable contract between client & algorithm

• Currently defined problems
  • Problem – basic, acyclic problem
  • CyclicProblem – for pipelined execution
  • SharedOperatorsProblem – limits #unit per operator type
  • ModuloProblem – for resource-constrained pipelined execution
  • ChainingProblem – models physical propagation delays

• Mix-and-match and extend to define your own problems!
  • Add properties, add/refine constraints
Schedulers

• Current goal: “Good enough” to bootstrap prototype flows
  • LP-based (using in-tree simplex solver) for all pre-defined problems
  • ILP-based (using external solver via OR-Tools): API demo
  • List-scheduler: API demo

• Problem models provide a consistent API to implement scheduling algorithms
  • Infrastructure is not limited to linear programming
  • Anything (satisfying the solution constraints) goes!
State and plans

• Available infrastructure in CIRCT
  • 5 problem models
  • Reference schedulers

• Current clients
  • End-to-end flow from C/PyTorch to SystemVerilog (circt-hls project)
  • Retiming irregularly-placed systolic arrays (WIP @ Microsoft)

• Future plans
  • Integrate into more synthesis flows and evolve infrastructure as needed
  • Design dialect to import/export scheduling problems
  • Port state-of-the-art algorithms to CIRCT
  • Can we share code with sibling projects or other parts of LLVM?
Thanks!

• Learn more: https://circt.llvm.org/docs/Scheduling/

• Get involved in CIRCT:
  • https://circt.llvm.org
  • ODM: Wednesdays @ 11am PT

• We thank Morten Borup Petersen, Stephen Neuendorffer, Aaron Landy, and the CIRCT community for their insightful discussions & contributions!