Artifact: Hyperblock Scheduling for Verified High-Level Synthesis

Getting started guide

- Try running and simulating Vericert on the benchmarks.

Step-by-step instructions

Coq formalisation

This section will give an overview of the Coq formalisation, and how it relates to the definitions present in the paper. An overview of the development is given first, followed by

First, the main difference between the paper and the formalisation is the naming of the intermediate languages. `RTLBlock` from the paper is named `GibleSeq` in the Coq formalisation, and `RTLPar` from the paper is named `GiblePar` in the Coq formalisation. In addition to that, `HTL` from the paper was renamed to `DHTL` in the Coq formalisation.

Using Fig. 1. from the paper as a general guide, the additions that were made to Vericert can be split into the following categories:

1. **`RTL`**: `RTL` is part of CompCert, the definition can be found in `lib/CompCert/backend/RTL.v`.
2. **`RTLBlock`**: RTLBlock is an intermediate language of basic blocks, with support for representing hyperblocks through predicated instructions. It is named `GibleSeq` in the Coq Formalisation. The base definition of the language can be found in `src/hls/Gible.v`, which contains definitions that are shared among other languages. Then, the specialised definition of `GibleSeq` can be found in `src/hls/GibleSeq.v`.
3. **Find BBs**: This transformation pass builds basic blocks from the CompCert `RTL` CFG. The files corresponding to this translation are the following:
    - `src/hls/GibleSeqgen.v`: This file contains the implementation of the basic block generation. It transforms an `RTL` program into a `GibleSeq` program, where no instructions are predicated. This transformation is mainly performed by an external function `partition` that generates the basic blocks, so this file only defines a validation algorithm used to check that the result of the external function was correct.
    - `src/hls/Partition.ml`: This file implements the unverified `partition` function that is later validated.

    - `src/hls/GibleSeqgenproof.v`: This file implements the proof of correctness of the basic block generation transformation, by showing that the validator will only accept transformations if these were in fact correct.

4. **If-conversion**: Next, the basic blocks are transformed into hyperblocks by if-conversion. If-conversion is split into three distinct phases:

    - `src/hls/CondElim.v` and `src/hls/CondElimproof.v`: These two files contain the implementation and proof of conditional elimination, which removes any branches from the basic blocks and replaces them by conditional goto instructions.
    - `src/hls/IfConversion.v` and `src/hls/IfConversionproof.v`: These two files implement the actual if-conversion algorithm by selecting goto instructions that should be replaced by the blocks they are pointing to. This translation pass is called multiple times.
    - `src/hls/DeadBlocks.v` and `src/hls/DeadBlocksproof.v`: These two files implement dead block elimination using a depth-first search algorithm, and removing any blocks that are not reachable from the entry point of the function.

5. **RTLPar**: RTLPar is the intermediate language that represents the result of the scheduling operation. It also contains hyperblocks, but contains a few more nested lists to represent the different ways in which instructions may have been scheduled. RTLPar is also based on `Gible.v`, and is then mainly implemented in `GiblePar.v`.
6. **Schedule**: The scheduling implementation is the core of the contribution.

    -

7. **FSM Generation**:
8. **Forward substitution**:

More detailed reproduction of experiments

Rebuilding the docker image

The docker image that was downloaded can be simply rebuilt from this `git` repository. To do so, use the following command

```
docker build --tag vericert/pldi2024 --file ./artifact/Dockerfile .
```

This can take around 30 mins because it will setup a fresh nix environment with all the necessary dependencies, and it will also download the Bambu 2023.1 AppImage.

Working without the docker environment

It should not be too difficult to work without the docker image, as it is only a light wrapper around setting up a `nix`. This should make it possible to reuse Vericert in new developments.

External dependencies that are needed and are not pulled in by the Docker image automatically:

- Bambu 2023.1 AppImage
- Optional: Xilinx Vivado 2023.2