HLCD Lab1：PDE Solving

> **DDL: 2025.10.21 23:59:59 (3 weeks)**

Please carefully read the `README` line by line.

Description

In this lab, we aim to design an efficient HLS implementation for 2D Jacobi iterative stencil operation, a common kernel in PDE solving. Given a 2D grid of size $M \times N$, the kernel is defined by the following equations:

$$u_{i,j}^{n+1} = w_1(u_{i-1,j}^n + u_{i+1,j}^n) + w_2(u_{i,j-1}^n + u_{i,j+1}^n) + w_3 u_{i,j}^n + c_{i,j}$$

where $0 \leq i < M$ and $0 \leq j < N$.

The boundary conditions are given by:

$$u_{i,j}^n = 0, \quad \text{if } i < 0 \vee i \geq M \vee j < 0 \vee j \geq N$$

Here, $u_{i,j}^n$ represents the PDE solution at grid point (i, j) for iteration n. The term $c_{i,j}$ is a constant value at grid point (i, j), and w_1, w_2, w_3 are constant weights. For this lab, we use a grid size 1024×1024 and focus on optimizing the computation **within a single iteration**.

You can find the example code in the `./src/top_tb.cpp` file.

You can complete this lab either on your local machine or on the server.

- **If you work locally:** You'll need to install **Vitis HLS 2021.1 beforehand**. Installation instructions can be found in file `vitis-hls-local-installation-guide.pdf`. Please note that Vitis HLS requires a **significant** amount of disk space (~140GB), so you should free up some space in advance.
- **If you work on the server:** The Vitis HLS 2021.1 environment is already set up.

We have the following **constraints**:

- Remain the file structure and names (`jacobi.h`, `jacobi.cpp`, `top_tb.cpp`) **unchanged**. You are only allowed to change the code in `jacobi.cpp`.
- Use `jacobi` as the kernel name.
- Use `m_axi` interface for `w1`, `w2`, `w3`, `u_prev`, `c`, `u_cur`, as the example code in `jacobi.cpp` does. **The data type should be `double` and you must not change it, but you can modify interface attributes.**

- Follow the annotations in `jacobi.cpp`.
- **Vitis version matters**. Different versions can affect the synthesis result. TA use the version **2021.1**, and we suggest to use this version.
- Set target period as **10ns**. **Slack and timing violation is allowed**.
- Flow target is **"Vivado IP Flow Target"**
- Choose part **"xc7z045fbg676-1"**.

Grading

TA gives a score board as shown below. For each row, if **both the latency and all resource metrics** meet the requirements, your design could get the score. Your final score is certainly the **highest** score of your design.

Score	Latency(cycles)	BRAM(%)	DSP(%)	FF(%)	LUT(%)
0	>10M	>100	>100	>100	>100
0-60*	10M-5.5M	100	100	100	100
60-85*	5.5M-1.2M	50	80	60	80
85-100*	1.2M-160K	30	60	30	60
100	<160K	15	50	20	50

Note: here, * means assign score **linearly** by latency. Resource metrics are measured in **percentages**.

Evaluation

TA provides a python script `test.py` for evaluation. It executes `script.tcl` to create the HLS project named `lab1_PDE`, add the codes and data under `./src`, and apply C Simulation and Synthesis. It will automatically fetch the C Simulation result and score the Synthesis result from the reports provides by the Vitis HLS tool.

For evaluating on your local platform:

1. Install Python3 and NumPy first
2. Configure Vitis HLS installation path in `test.py` (if you use Windows, the path should like "D:\\aa\\bb")
3. run `python3 test.py` in your current directory

For evaluating on the server: just run `python3 test.py`

Submission

In this lab, you have to submit your homework **on the server**, including:

1. The **source code** of your kernel
2. One **report** (`.pdf`) that describes your optimization techniques. The report can be written in either Chinese or English, and is **at most 3 pages**.
   Please put all the things in `~/hw/lab1_PDE` in our server, the folder should look like:

```
lab1_PDE
|-- src
|   |-- jacobi.cpp
|   |-- jacobi.h
|   `-- top_tb.cpp
|-- report.pdf
`-- test.py
```

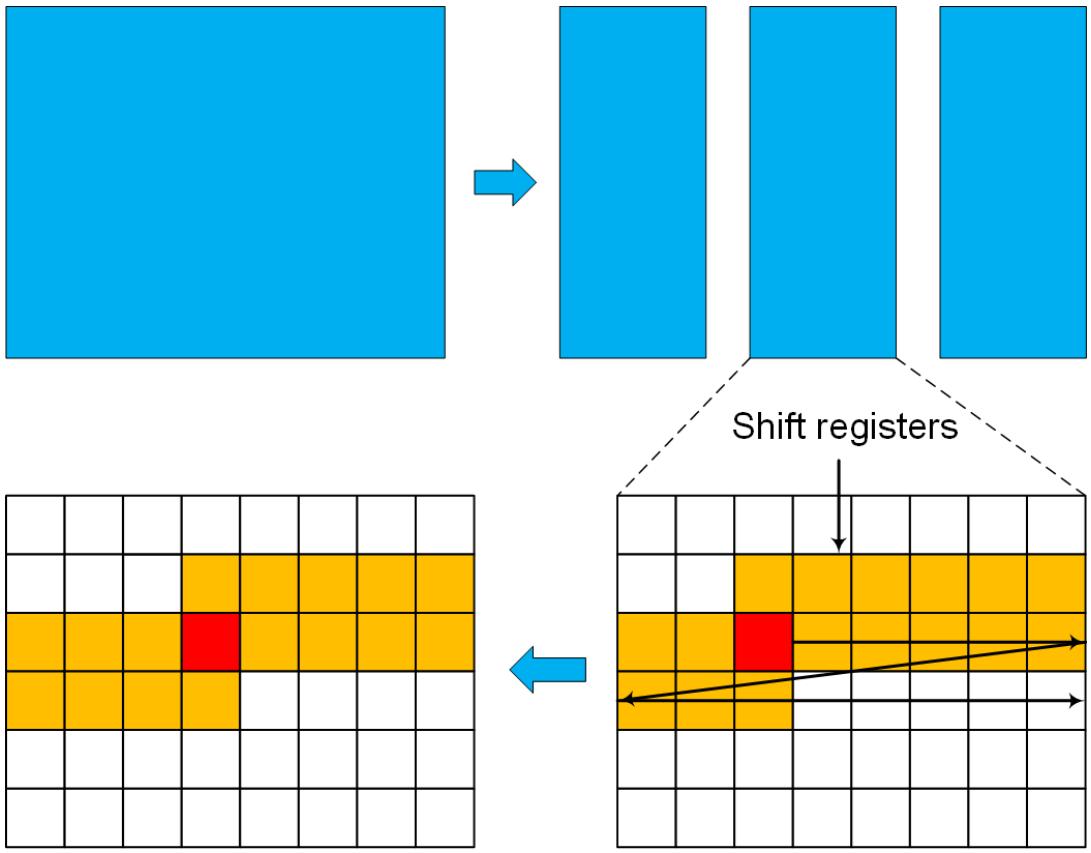
Suggestions

TA gives the following suggestions and hints:

- Get familiar with the HLS design methodology. Our course gives a brief and practical introduction. You may also refer to Vitis High-Level Synthesis User Guide (UG1399) (Especially the first two chapters) for more information.
- The default optimization applied by Vitis HLS may be **harmful**. You may disable them first, for example, `#pragma HLS pipeline off` .
- The following optimization techniques may help:
  - Loop unrolling/pipelining
  - Array partitioning
  - Compositive data type (Struct) and data aggregation
  - Dataflow (you don't need to design a streaming architecture for this project).

Hints

- **To achieve the score 60**, you only need to implement a basic kernel with only the reuse of constant weights w. However, the pipeline efficiency is low (II=5).
- **To achieve the score 85**, you also need to reuse the input data $u_{i,j}$. One solution is to partition the grid into tiles and reuse the data through shift registers as illustrated in the following figure. After data reuse, you can achieve full pipeline of the kernel (II=1).



- **To achieve the full score**, you will need to fully utilize the 512 bitwidth of `m_axi` interface. This allows you to fetch 8 double-precision data points from memory in a single cycle (please refer to the Vitis user guide for more details), and thus you can perform 8 computations per cycle. To fully pipeline the 8 computing units, you can refer to the hardware design in FDMAX paper.