Dense SLAM Report

Jonathan Lam, Hanyang Xu

Dense SLAM (Simultaneous Localization And Mapping) makes 3D reconstruction of the environments. It requires high computational power while having vast potential for parallel optimization. In this project, we implemented one of the major functions, ICP (Iterative Closest Point) algorithm, in Dense SLAM framework InfiniTAM in Vivado HLS and optimize its performance on FPGA.

Introduction:

SLAM (Simultaneous Localization and Mapping) is a technique that tracks and reconstructs a live scene based on some moving sensor, usually a 3D camera. A SLAM framework usually involves 1) Building a 3D object representation(model) 2) Tracking camera's movement and pose 3) Extracting and Combining new Image information into existing model. [1]

InfiniTAM SLAM framework is open source, multi-platform framework for real-time, large-scale depth fusion and tracking developed at University of Oxford. It has four major components: 1) a global Map that represents 3D objects 2) Depth Fusion: Update the depth map with new camera movement 3) Ray Casting: construct a 3D model from a 2D view. 4) Tracking: tracking the camera movement using ICP algorithm. The ICP algorithm finds the optimal alignment between the input depth map and the 2D project of the current model and the alignment transformation represents the camera movement. [2]

The ICP algorithm is a major method to align 3D models based purely on geometry. It starts with two models and an initial guess of their alignment/transformation, and iteratively updates this transformation based on a lost function that depends on the closed distance between points. [3] In InfiniTAM framework, the algorithm is implemented as follow [4]:

- Render a map V of surface points and a map N of surface normals from the viewpoint of an initial guess
- Project all points p from the depth image onto points $\bar{p}$ in V and N and compute their distances from the planar approximation of the surface
- Find R and t minimizing of the sum of the squared distances by solving linear equation system
- Iterate the previous two steps until convergence

Objectives:

The objectives for our project, as described in previous reports, are:

- Objective 1: Run the infiniTAM software version on PC using sample data.

- Objectives 2: Understand the implementation of InfiniTAM sub-functions and and choose one sub-function to implement in HLS
- Objectives 3: Get the software version output “ground truth” and store it in file for comparing output. Implement testbench in HLS.
- Objectives 4: Implement the chosen sub-function on HLS and test it against ground truth
- Objectives 5: Implement functional sub-function on PYNQ board

Implementations:

We choose the ICP algorithm as the sub-function to implement in HLS after we study the InfiniTAM framework. We choose the ICP algorithm for two reasons: 1) It is an operation on every pixel of the input model which gives us the maximum parallelization possibilities 2) The inputs/outputs are clearly defined and bounded which allows us to compare our implementation with the ground truth.

ICP Algorithm:

When implementing just the ICP submodule of InfiniTAM, we analyzed, copied and rewrote the ICP kernel from *ITMSceneReconstructionEngine_OpenCL.cl* to compile within Vivado HLS. We had to create our own header in *icp.h* that outlines all different float, int and custom data structures used in the ICP kernel. This had to be done since we had no access to any vector libraries in writing the HLS variant and we also had to define our own version of dot products. Within the main ICP function, all instances of vectors had to be replaced and had to be manually assigned their individual values. One of the biggest differences in this version was the necessity to write the code such that, at compile time, the sizes of certain elements (depth, normalsMap, and pointsMap) were known. These arrays were sized to the maximum possible value they can have. This is set with variables such as MAX_IMG_X and MAX_IMG_Y within the header. Since viewImageSize contains the correct image size, its values are used to index into depth. The same applies to normalsMap and pointsMap with sceneImageSize.

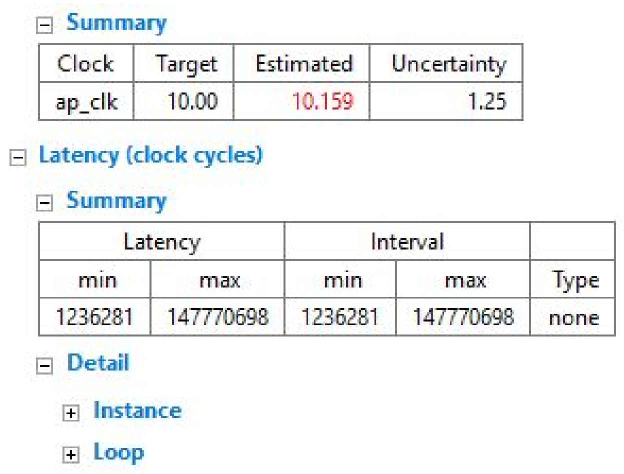
Testbench:

The Testbench is implemented after the input and outputs are defined by ICP function. The *ITMDepthTracker_CPU.cpp* in the original implementation is analyzed and modified to produce the inputs and golden outputs. Larger inputs are outputted to files: *depth.txt, normalsMap.txt* and *pointsMap.txt*, while shorter inputs/outputs are put together in *frame_zero_data.txt.* The testbench then constructs inputs from files, calls ICP function and compares outputs.

Results:

It should be noted that to obtain the maximum latency, we had to switch out certain loops.The most notable one is that instead of looping viewImageSize.x and viewImageSize.y we use their maximum possible values with the previously mentioned MAX_IMG_X and MAX_IMG_Y. If we were to keep the use of viewImageSize, sizes are unknown at compile time so we would not have the latency of our implementation.

Current Implementation



⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.159	1.25

⊟ Latency (clock cycles)

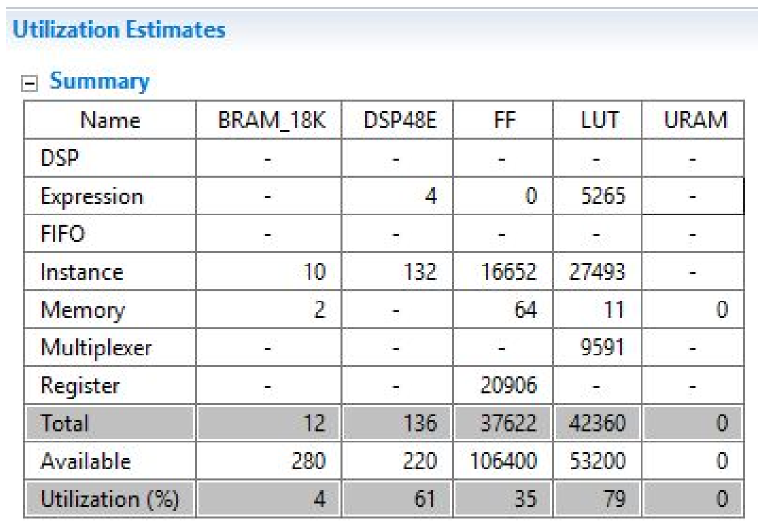
⊟ Summary

Latency		Interval		
min	max	min	max	Type
1236281	147770698	1236281	147770698	none

⊟ Detail

⊞ Instance

⊞ Loop



Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	4	0	5265	-
FIFO	-	-	-	-	-
Instance	10	132	16652	27493	-
Memory	2	-	64	11	0
Multiplexer	-	-	-	9591	-
Register	-	-	20906	-	-
Total	12	136	37622	42360	0
Available	280	220	106400	53200	0
Utilization (%)	4	61	35	79	0

It should be noted that the current implementation is untested due to a weird error we encountered. Despite being what it declares itself as a segfault, it occurs when no memory access should be occurring. Since in the algorithm, whenever a depth of 0 is read nothing occurs. But a segfault occurs towards the end of the program despite no memory access occuring. The index where it is occuring shifts whenever we add more debug statements. Hence why it’s a strange error.

Conclusion:

For this project, we ran a software implementation of ICP and analyzed it. We pulled input and output values to test against when recreating the submodule in Vivado HLS. We were able to get the submodule to synthesize within Vivado HLS but were unable to go any further. When doing initial testing to compare outputs, we were unable to correct an error that we experienced. Any further optimizations were halted in our attempt to fix it. When inspecting all memory accesses, it all seems to fall within the allocated spaces we have created. At this moment we can confirm that this implementation is able to fit onto a Pynq board but it currently does not meet a target clock of 10ns. More time was needed to analyze the error in further depth.

Source:

[1] R. A. Newcombe *et al*., "KinectFusion: Real-time dense surface mapping and tracking," *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, Basel, 2011, pp. 127-136.
[2] Q. Gautier, A. Althoff and R. Kastner, "FPGA Architectures for Real-time Dense SLAM," *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, New York, NY, USA, 2019, pp. 83-90.
[3] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, Quebec City, Quebec, Canada, 2001, pp. 145-152.
[4] Prisacariu, Victor & Kähler, Olaf & Golodetz, Stuart & Sapienza, Michael & Cavallari, Tommaso & Torr, Philip & Murray, David. (2017). InfiniTAM v3: A Framework for Large-Scale 3D Reconstruction with Loop Closure.