Computer Systems Lab

Comparative Performance Analysis of Vulkan Implementations of Computational Applications

Maria Rafaela Gkeka,
Nikolaos Bellas, Christos D. Antonopoulos

ECE Department
University of Thessaly, Greece

Outline

→ Motivation
- Background
- Local Laplacian Filters (LLF) algorithm
- LLF Implementations and Experimental Evaluation
- VO KinectFusion algorithm for SLAM (Simultaneous Localization and Mapping)
- VO KinectFusion Implementations and Experimental Evaluation
- Discussion

Motivation

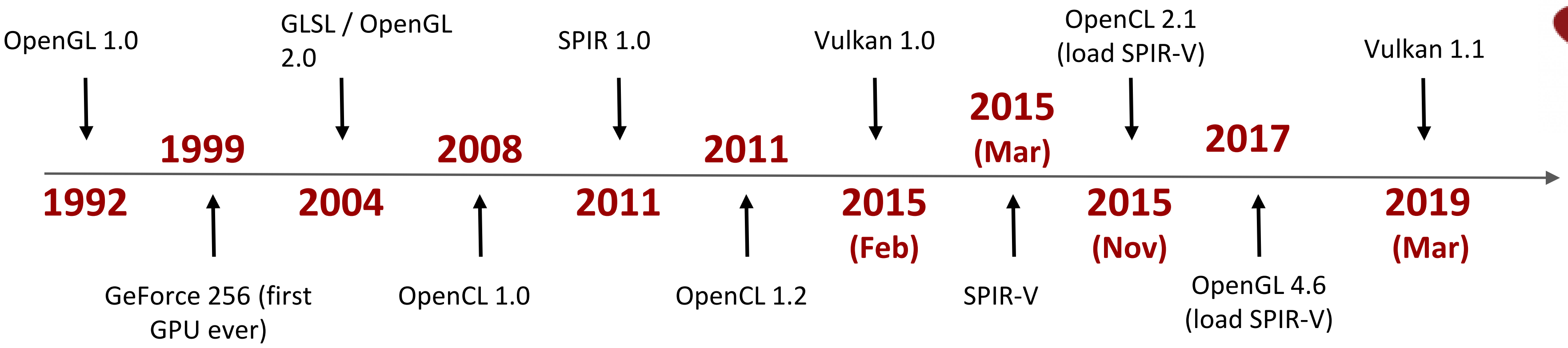
- GPUs widely used as compute and graphics accelerators
  - Multiple APIs used to program them (OpenCL, OpenGL, CUDA, Vulkan, …)
- Vulkan is a new API aiming (among others) at integrating compute and graphics pipelines
  - As of today, almost exclusively used for 3D graphics
- There is a need to better understand Vulkan performance implications in realistic compute applications

Outline

- Motivation
- → Background
- Local Laplacian Filters (LLF) algorithm
- LLF Implementations and Experimental Evaluation
- KinectFusion algorithm for SLAM (Simultaneous Localization and Mapping)
- KinectFusion Implementations and Experimental Evaluation
- Discussion

Background

- Vulkan: modern low-overhead cross-platform 3D graphics and compute API
  - targets high-performance real time 3D graphics applications such as video games
  - uses the Khronos SPIR-V intermediate representation with native support for shader features



Outline

- Motivation
- Background

→ Local Laplacian Filters (LLF) algorithm

- LLF Implementations and Experimental Evaluation
- KinectFusion algorithm for SLAM (Simultaneous Localization and Mapping)
- KinectFusion Implementations and Experimental Evaluation
- Discussion

Local Laplacian Filter

- Given an input image, the algorithm applies detail or tone enhancements.
- Edge-preserving image processing algorithm based on the direct manipulation of Gaussian/Laplacian pyramids. [1]

input image

Output image with $(\alpha,\beta,\sigma_r)=(0.25,1,0.4)$

Output image with $(\alpha,\beta,\sigma_r)=(4,1,0.2)$

[1] Sylvain Paris et al. Local Laplacian Filters: Edge-aware Image Processing with a Laplacian Pyramid *ACM Trans. Graph. 30.4 (2011)*

LLF is a pyramid-based algorithm

Gaussian pyramid

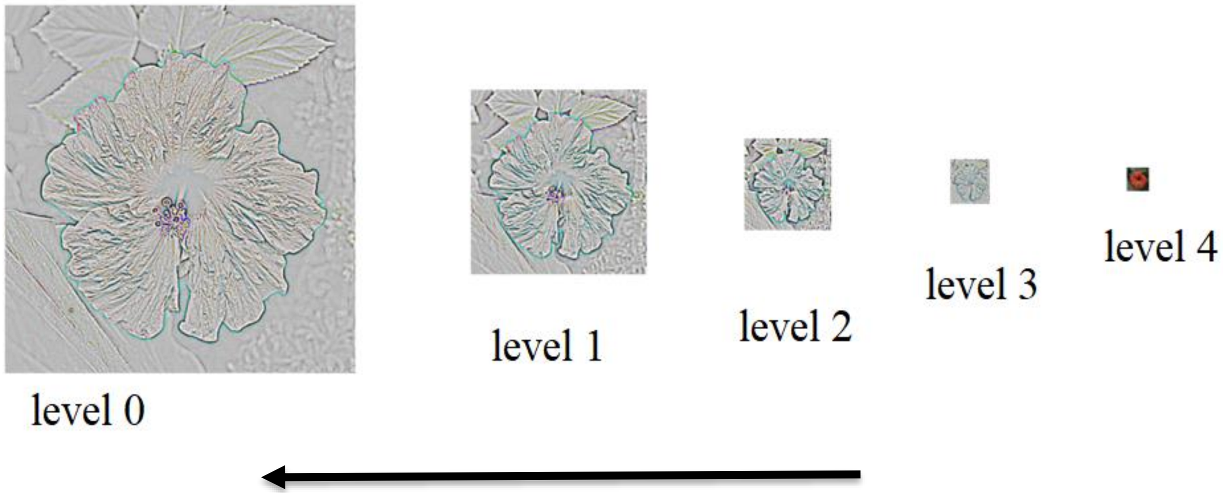
L_{i+1} image generated by blurring (5-by-5 Gaussian filter) the L_i level image and downsample by 2

$$\begin{pmatrix} 0.0025 & 0.0125 & 0.02 & 0.0125 & 0.0025 \\ 0.0125 & 0.0625 & 0.1 & 0.0625 & 0.0125 \\ 0.02 & 0.1 & 0.16 & 0.1 & 0.02 \\ 0.0125 & 0.0625 & 0.1 & 0.0625 & 0.0125 \\ 0.0025 & 0.0125 & 0.02 & 0.0125 & 0.0025 \end{pmatrix}$$

Laplacian pyramid

Saves the error between Gaussian images at intermediate levels.

The algorithm adds in each level image the expanded image of the lower level.



LLF algorithm

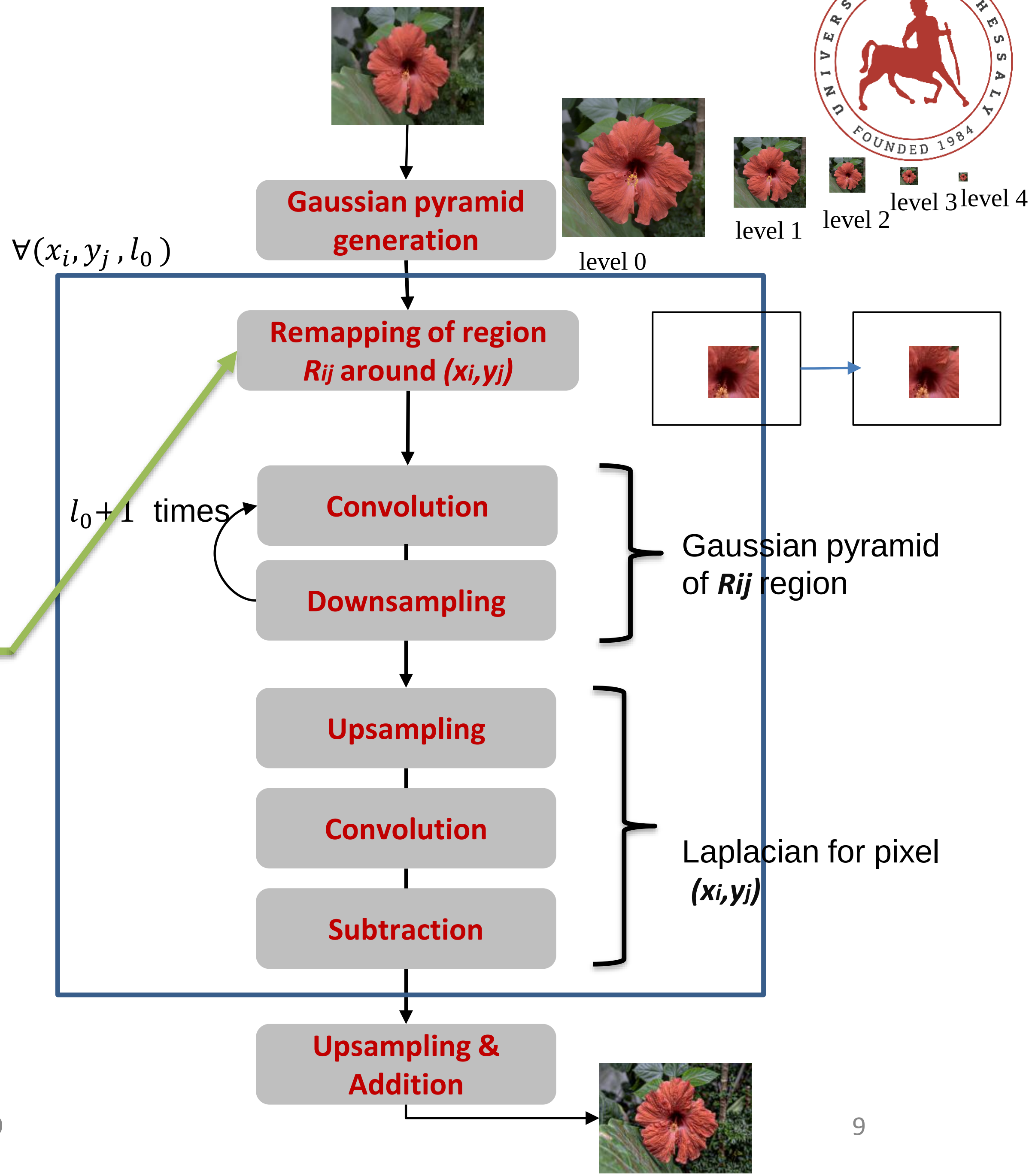
Remapping

Determines the kind (detail, edge) of each pixel and manipulates the neighbor region.

Creates a new sub-image for each Gaussian pyramid image pixel (g_0).

$$r_d(i) = g_0 + unit(i - g_0)\sigma_r f_d(||i - g_0||/\sigma_r)$$

$$r_e(i) = g_0 + unit(i - g_0)[f_e(||i - g_0|| - \sigma_r) + \sigma_r)$$



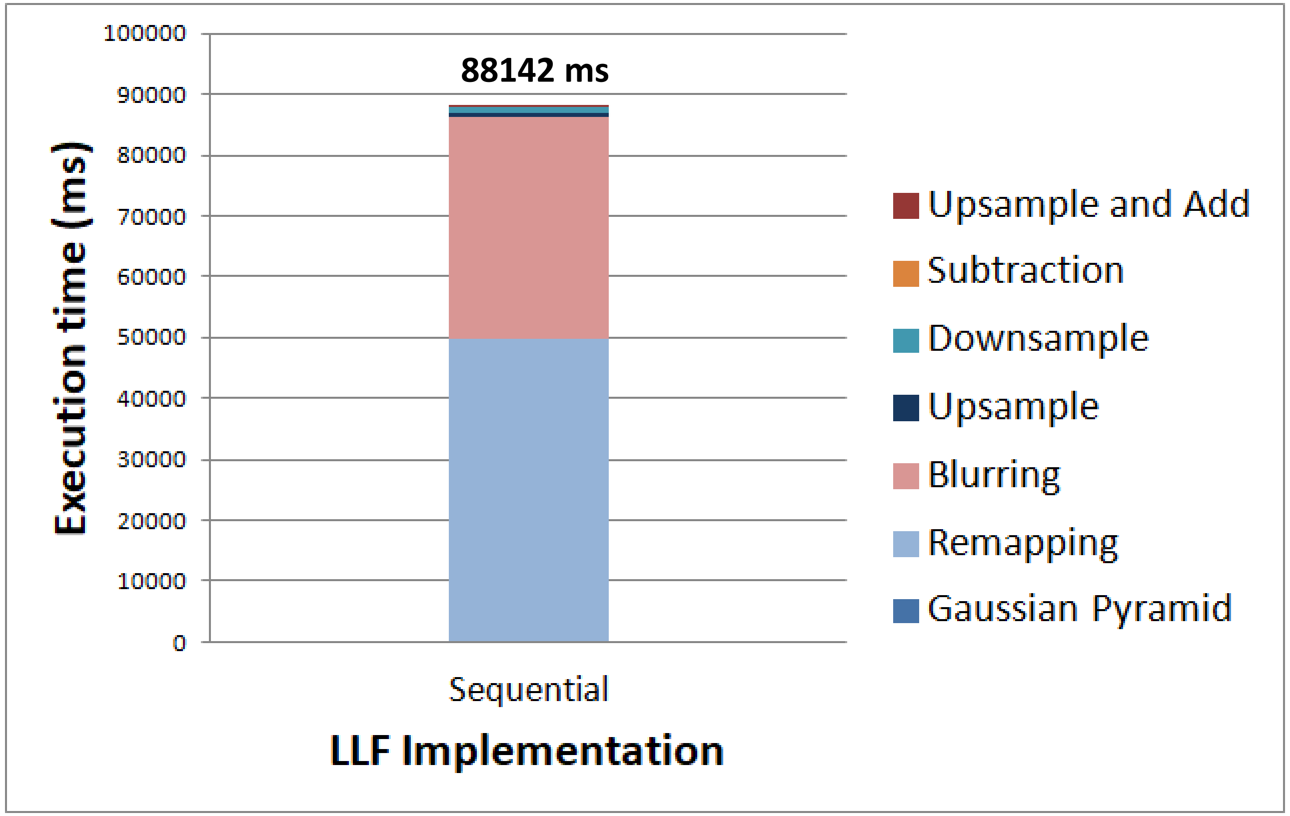
Outline

- Motivation
- Background
- Local Laplacian Filters (LLF) algorithm

→ LLF Implementations and Experimental Evaluation

- KinectFusion algorithm for SLAM (Simultaneous Localization and Mapping)
- KinectFusion Implementations and Experimental Evaluation
- Discussion

LLF Sequential Implementation



Intel® Core i7-4820K @3.70GHz, 16GB DRAM

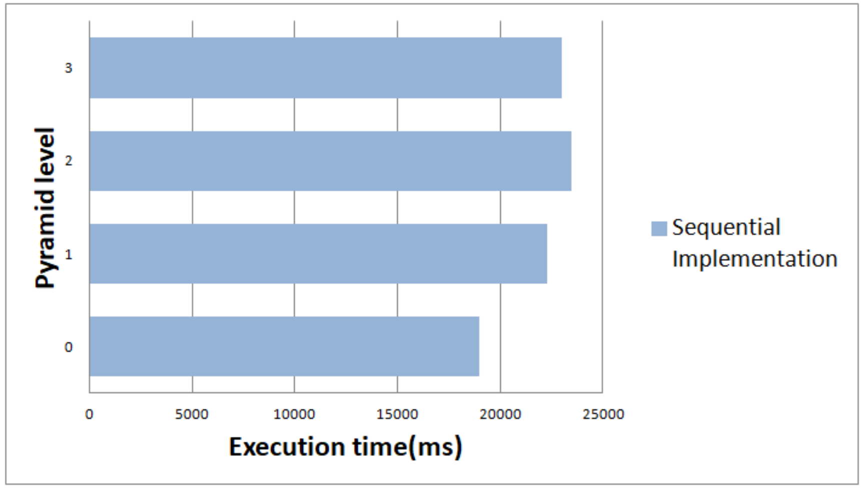
- Single threaded C code
- O3 optimizations

800x533 input image

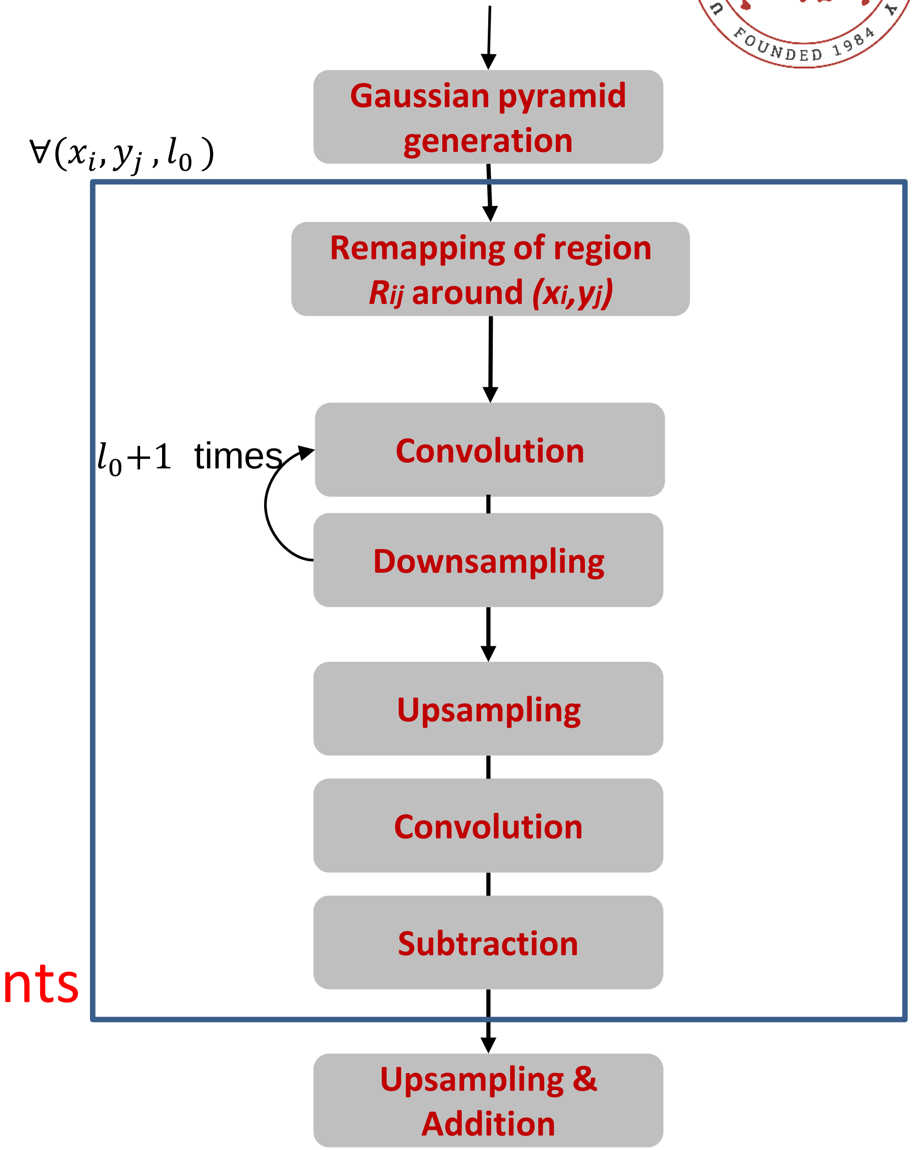
Where is parallelism?

+ Loop iterations are **independent**

+ Workload is **balanced** across pyramid levels



– But, parallel execution has very high memory requirements to store intermediate images



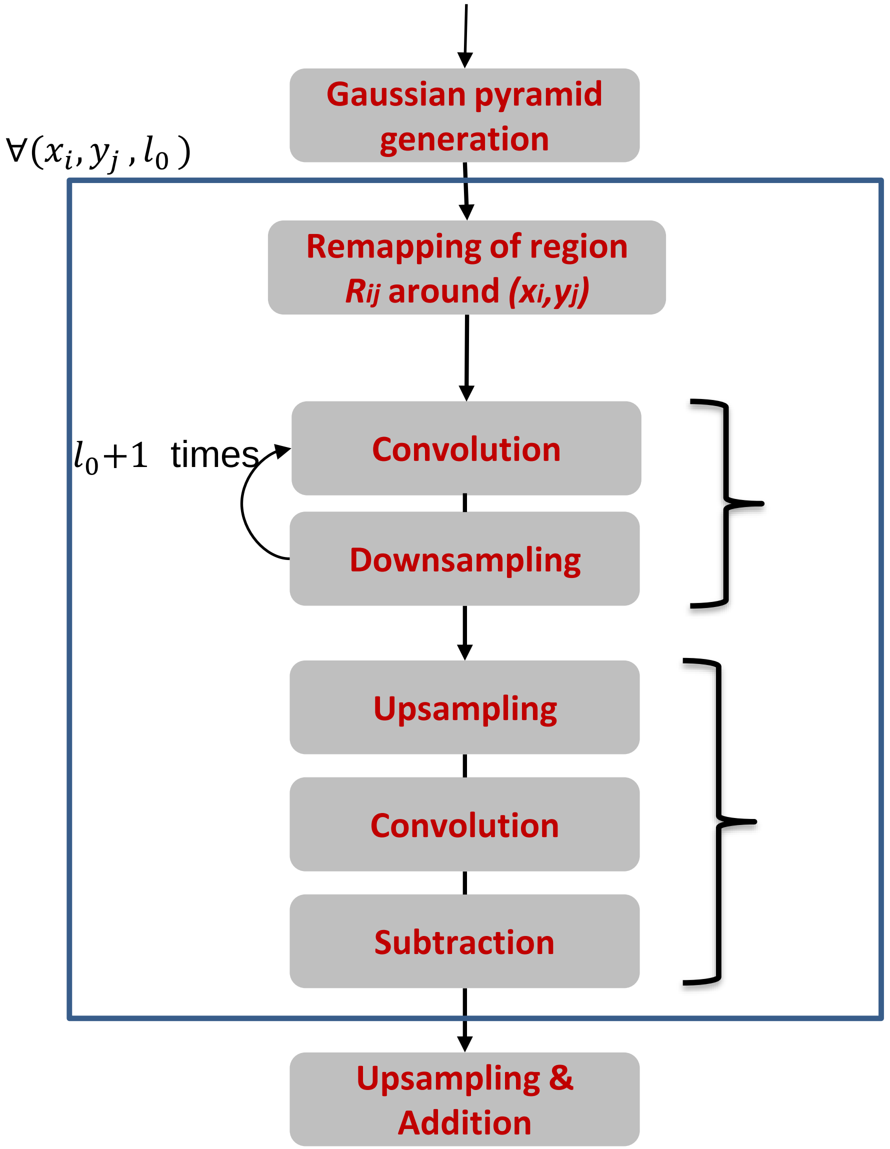
Data parallel execution

Five OpenCL kernels executing in sequence.

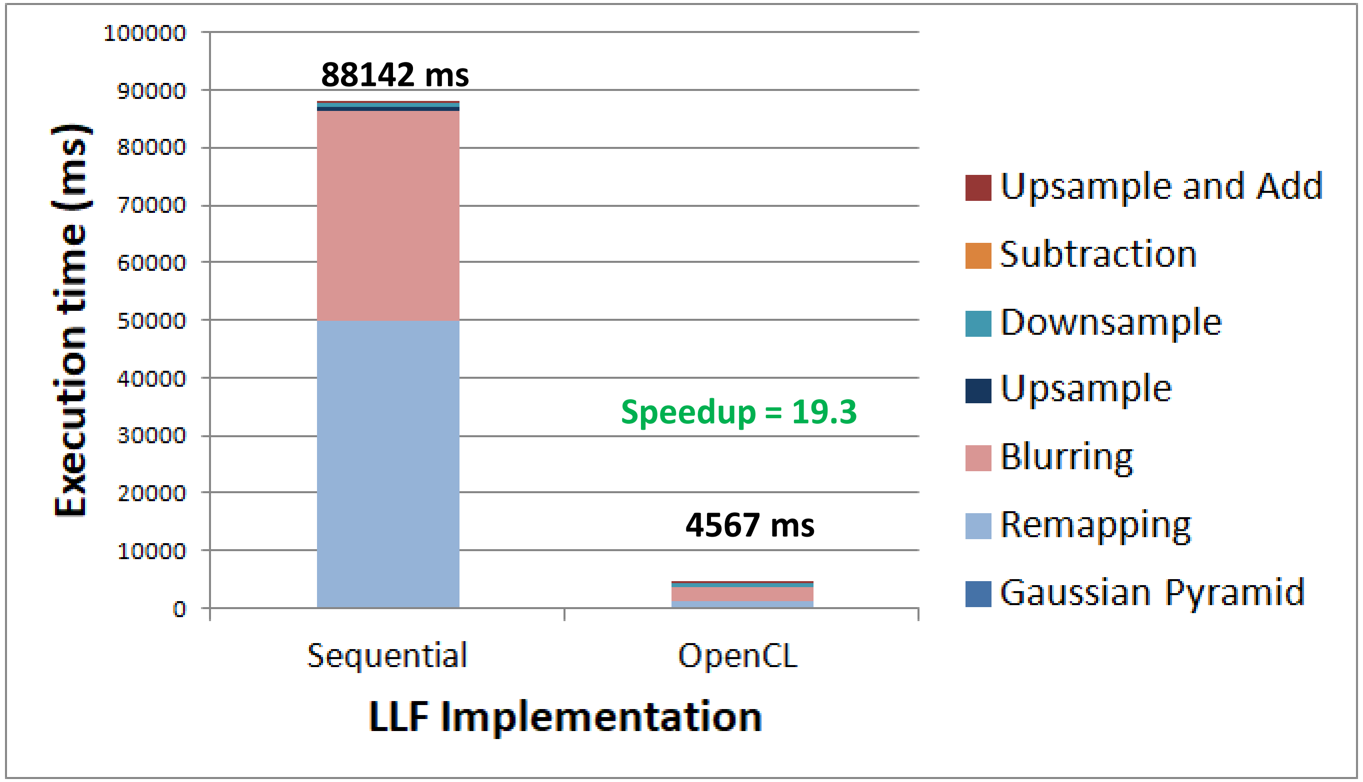
- Remapping
- Convolution (blurring)
- Downsampling
- Upsampling
- Subtraction

Grid: processing all pixels of a single line

Work-items: processing single pixel of the sub-image



Optimized OpenCL Implementation

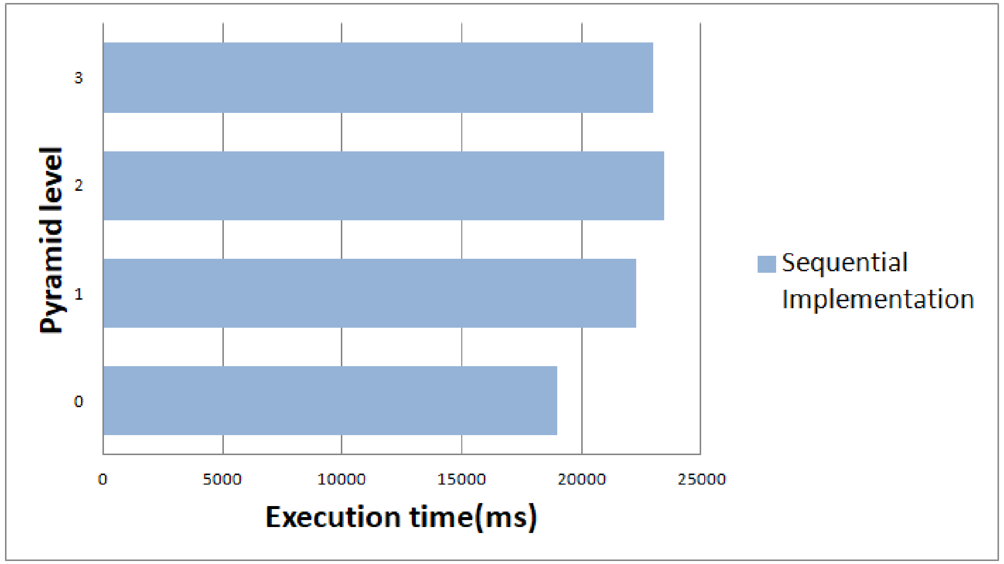


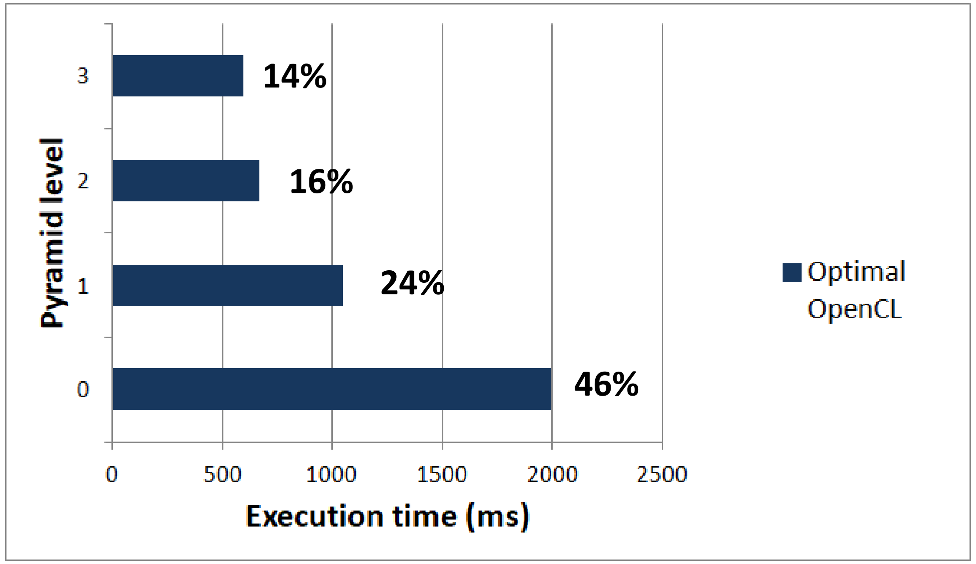
nVIDIA GeForce GTX 770 @1.046 GHz, 1536 SP cores

2 GB GDDR5

OpenCL 1.2

Execution time per pyramid Level





The ET of smaller levels increases because of the bigger region used for calculations of one pixel.

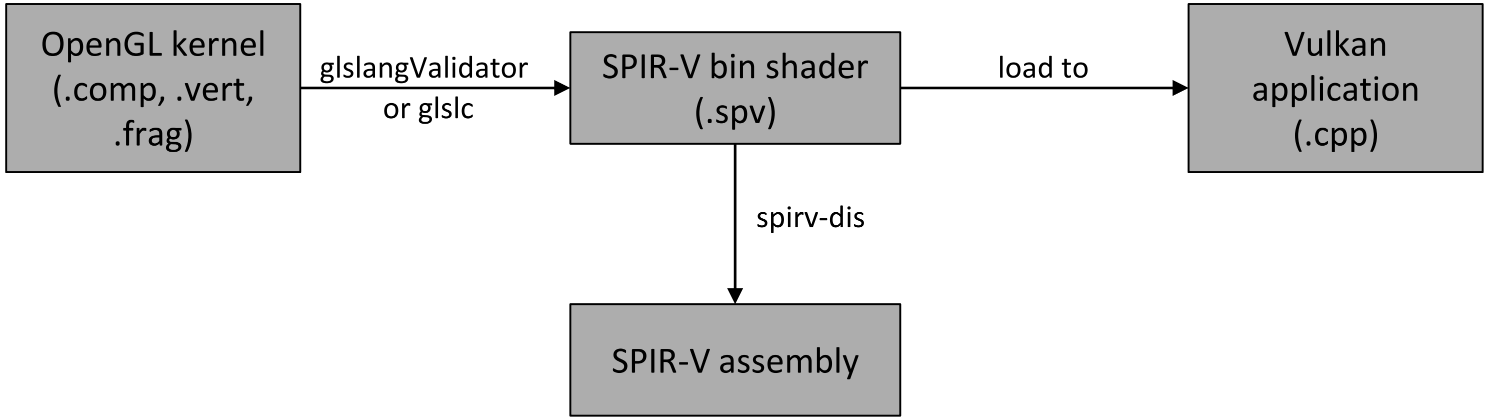
Parallel calculations of each pixel. The ET of smaller levels decreases because depends only on pyramid image size.

LLF Vulkan Implementation and Optimization

In Vulkan API all shaders (kernels) are presented in SPIR-V format. SPIR-V is a binary intermediate representation for graphical shaders and compute kernels.

In our implementation we use the glslangValidator for the compute shaders compilation.

Vulkan API Version: 1.0.61



Baseline Vulkan Implementation (v1)

Create a pipeline for each shader

Choose Memory Heap and Memory Type

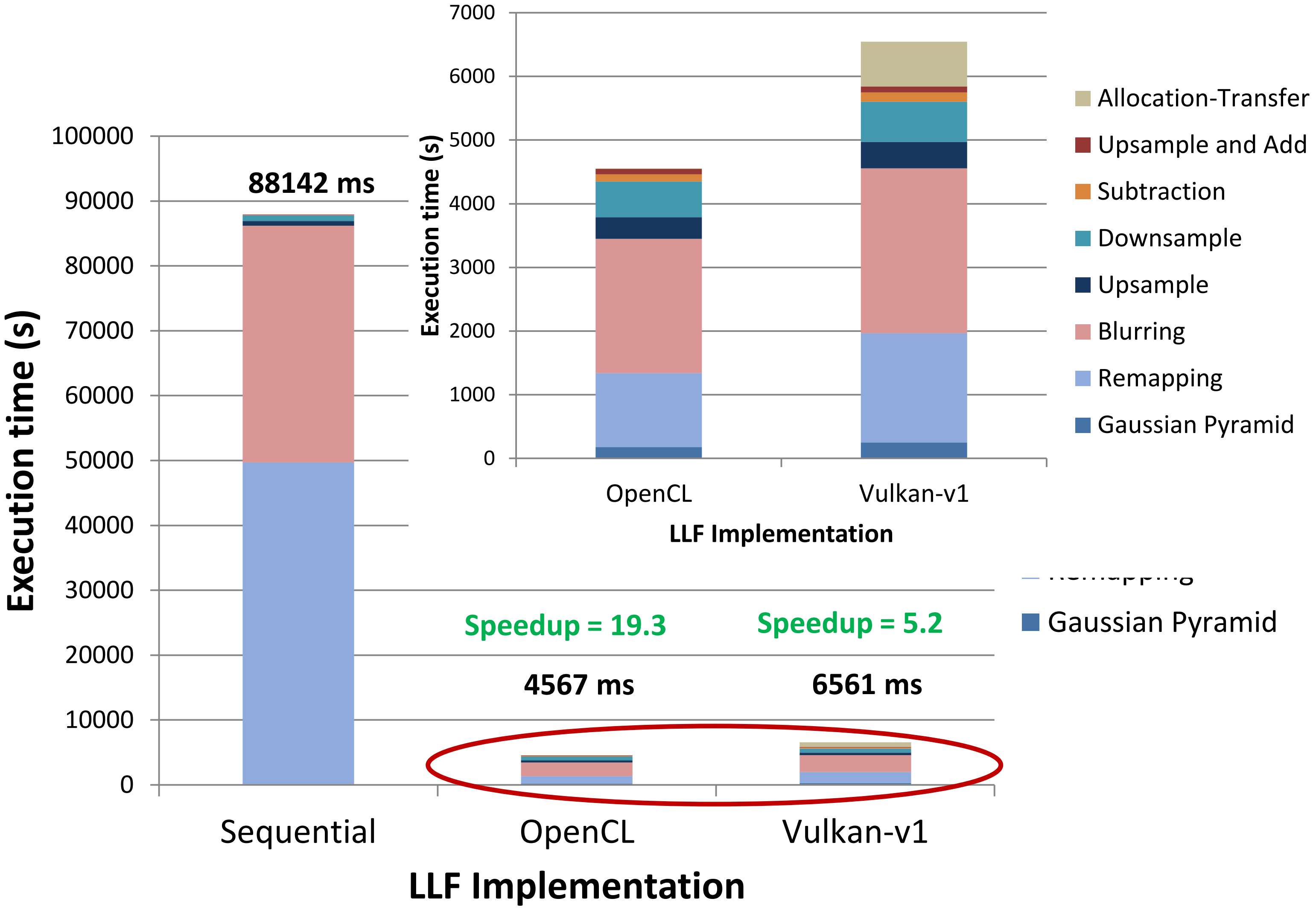
- Host or Device visible
- Allocate buffers in memory

Execute each shader

- Add each pipeline to a command buffer
- Submit the command buffer to execution queue

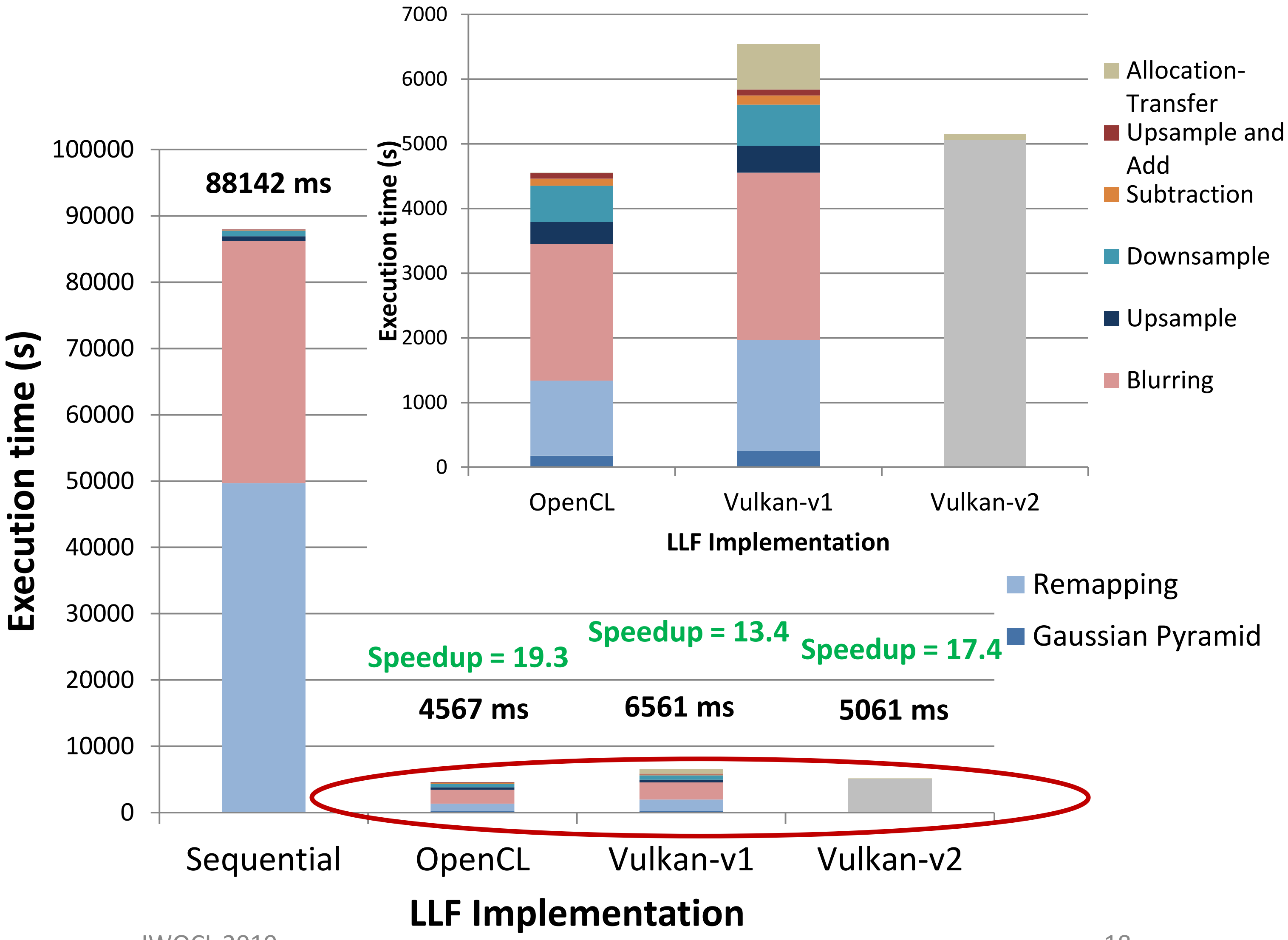
SPIR-V does not support parameterizable work-group size

Vulkan-v1: different kernel versions for each work-group size



Command Buffer optimization (v2)

- We can record the work of all iterations in one command buffer and synchronize using memory barriers between iterations.
- Lower overhead due to kernel launching compared with OpenCL/OpenGL (and CUDA)



Outline

- Motivation
- Background
- Local Laplacian Filters (LLF) algorithm
- LLF Implementations and Experimental Evaluation

→ KinectFusion algorithm for SLAM (Simultaneous Localization and Mapping)

- KinectFusion Implementations and Experimental Evaluation
- Discussion

Simultaneous Localization and Mapping (SLAM)

SLAM is used in robotics for autonomous movement:

- Dynamically building a map of the environment (mapping)
- Navigating this environment using the map while keeping track of the robot's relative position and orientation

- Used in many real systems and applications
  - Mobile robotics
  - Driverless cars
- Many algorithms and implementations

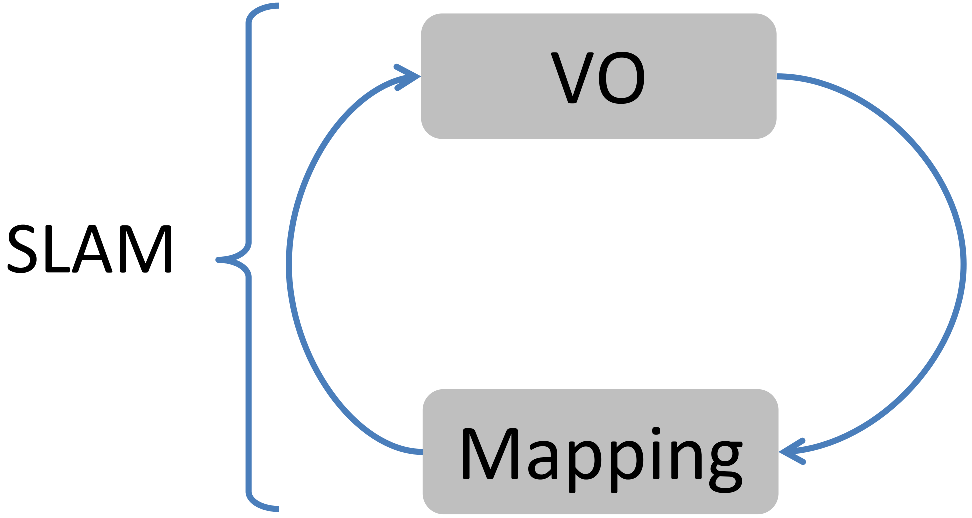
  → KinectFusion implementation in C++, OpenMP, OpenCL and CUDA [2]

[2] L.Nardi et al. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM *ICRA 2015*

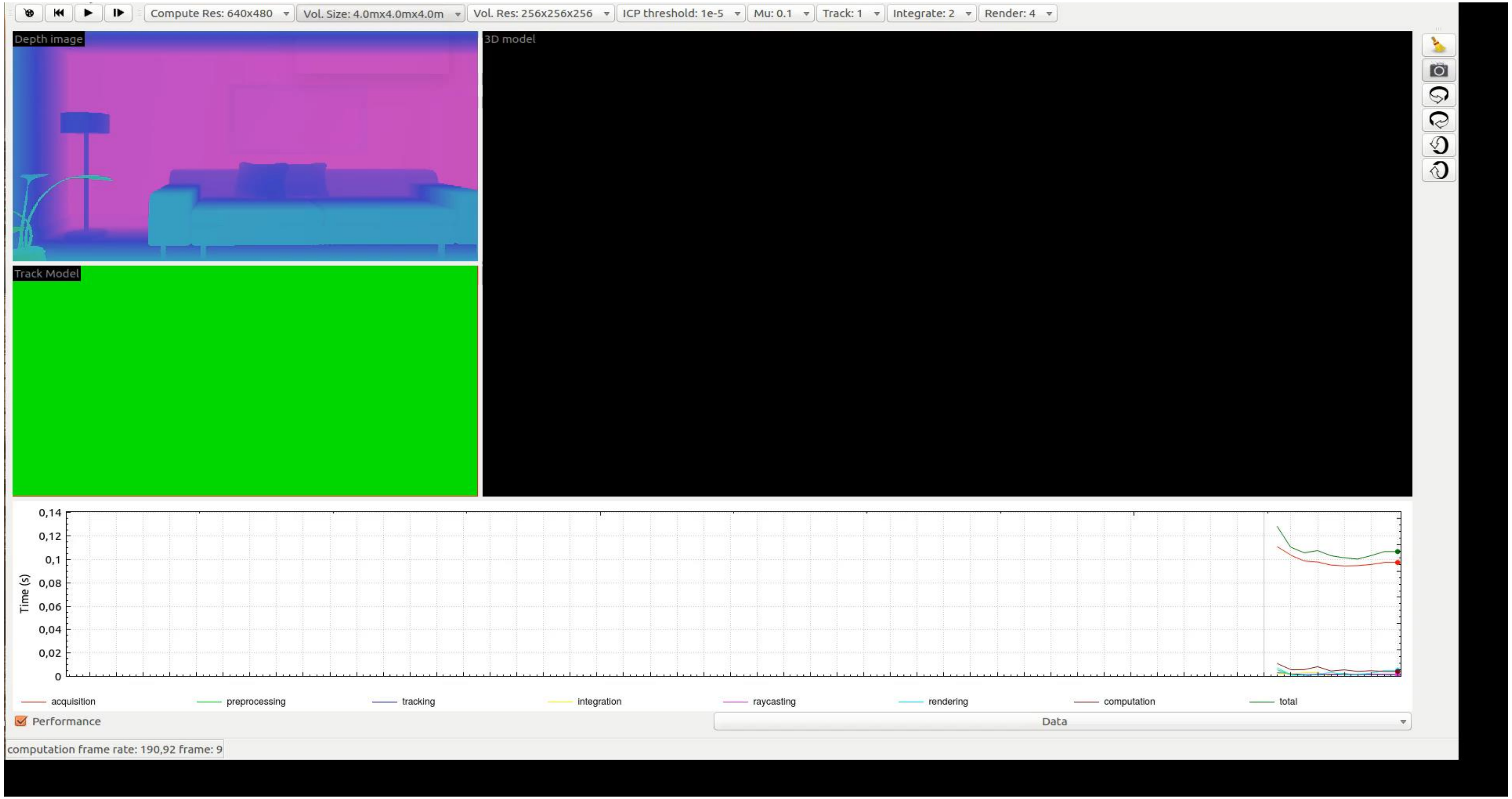
Visual Odometry (VO) & Mapping

- **VO**: the robot localize itself in its environment based on the constructed maps
  - without any human input

- **Mapping**: build the map of the environment based on sensory information
  - e.g. RGB-D cameras
  - Lidar



KinectFusion



Outline

- Motivation
- Background
- Local Laplacian Filters (LLF) algorithm
- LLF Implementations and Experimental Evaluation
- KinectFusion algorithm for SLAM (Simultaneous Localization and Mapping)

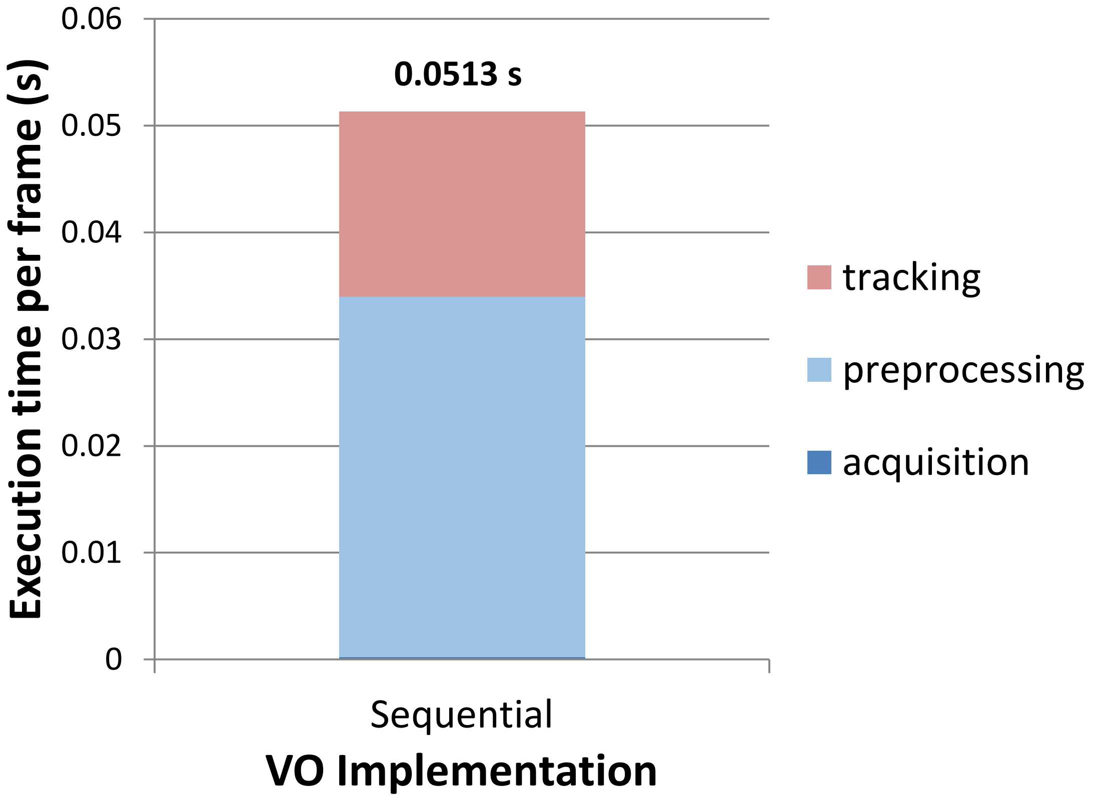
→ KinectFusion Implementations and Experimental Evaluation

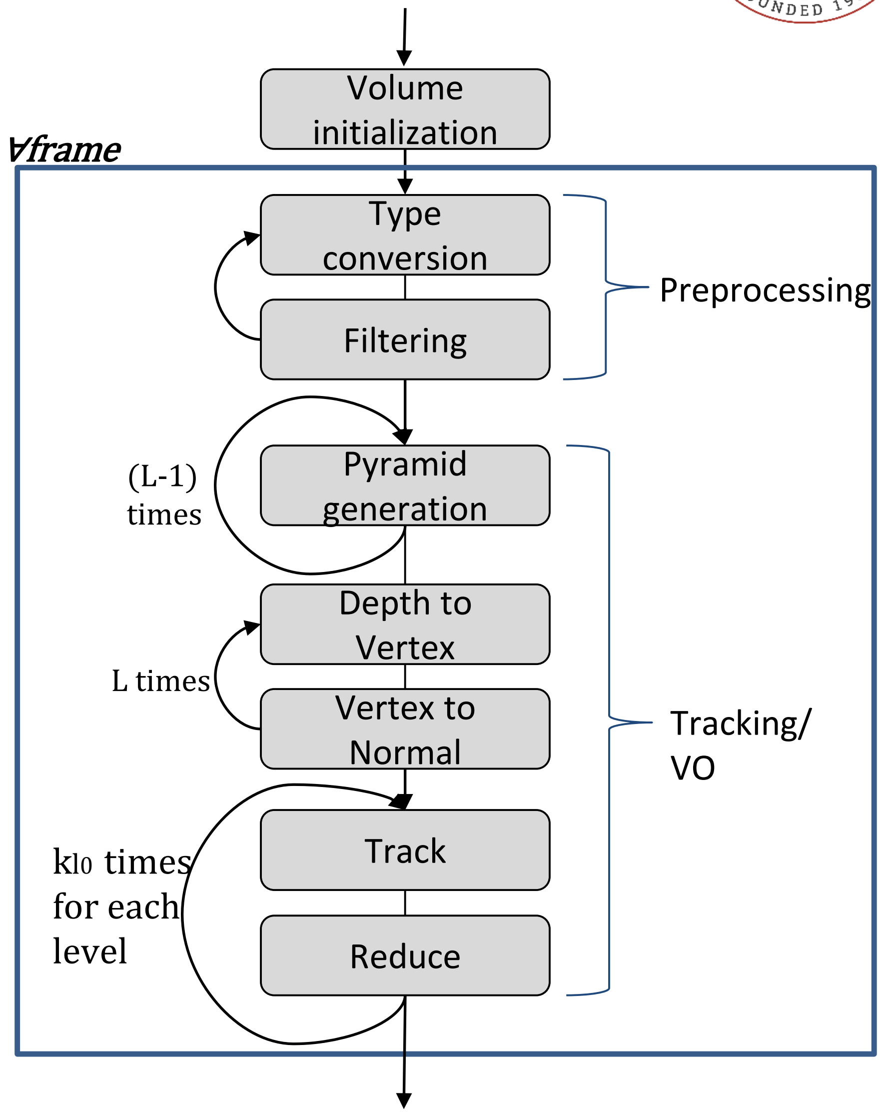
- Discussion

KinectFusion Sequential Implementation

Input

Video stream of 882 depth frames
VGA resolution (640x480)





KinectFusion OpenCL Implementation

OpenCL-v1: OpenCL compiler optimizations disabled

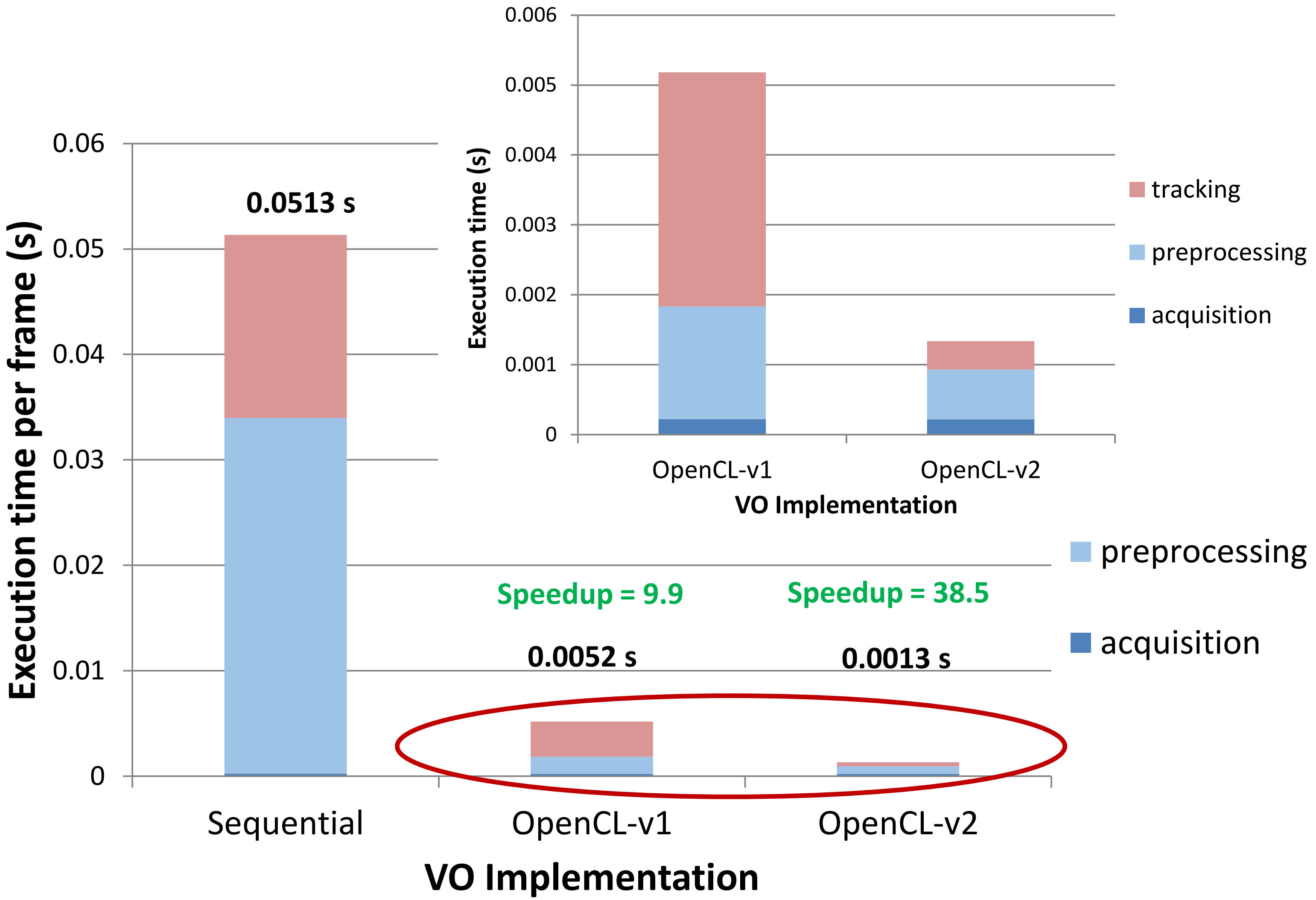
- *"-cl-opt-disable"*

OpenCL-v2: Default OpenCL compiler optimizations

nVIDIA GeForce GTX 770 @1.046 GHz, 1536 SP cores

2 GB GDDR5

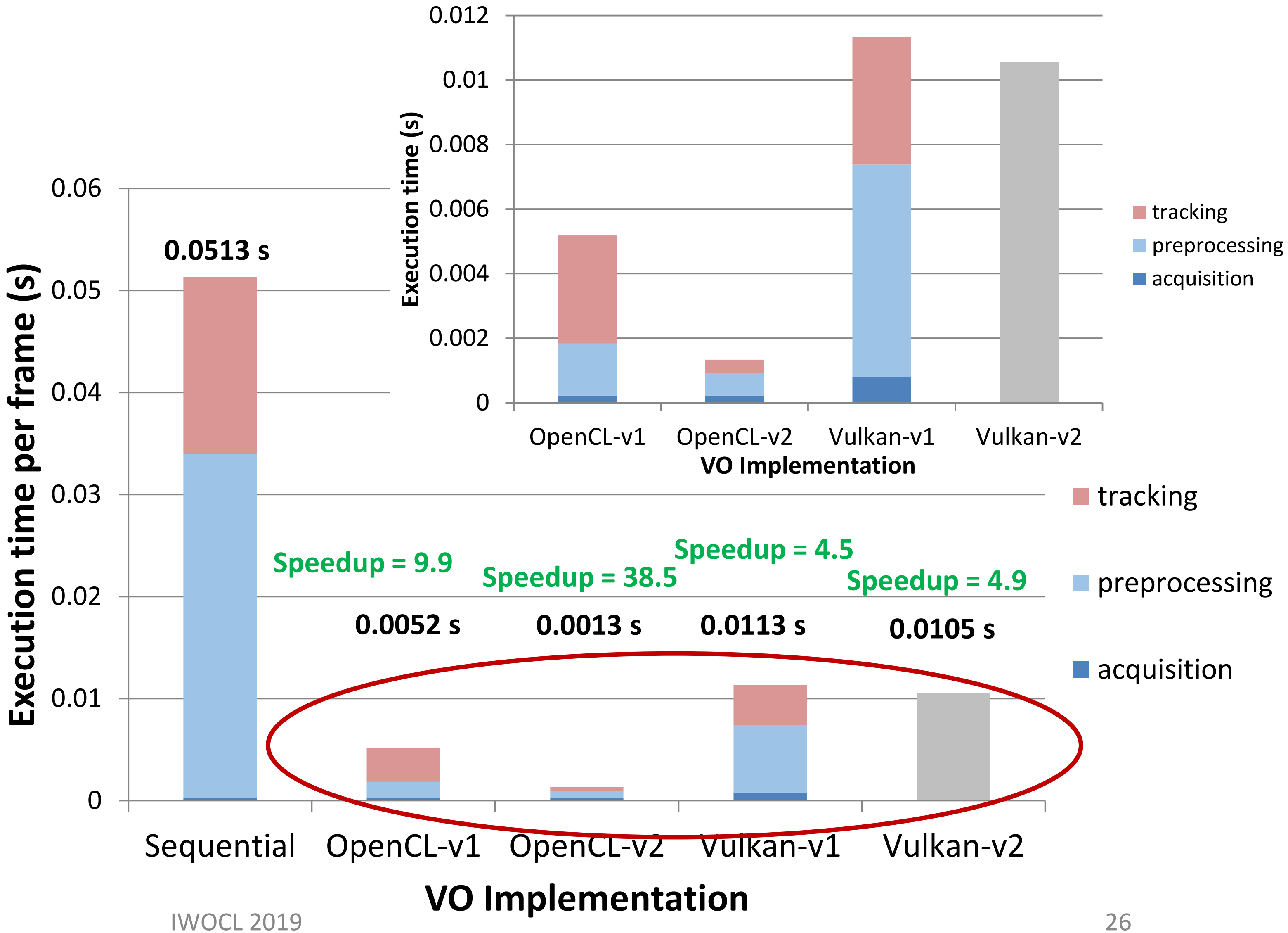
OpenCL 1.2



KinectFusion Vulkan Implementation

Vulkan-v1: Initial implementation

Vulkan-v2: Command buffers optimization



Outline

- Motivation
- Background
- Local Laplacian Filters (LLF) algorithm
- LLF Implementations and Experimental Evaluation
- KinectFusion algorithm for SLAM (Simultaneous Localization and Mapping)
- KinectFusion Implementations and Experimental Evaluation

→ Discussion

Discussion (I)

Vulkan has widely different behavior in the two apps.

Vulkan in LLF is almost as fast as OpenCL in LLF

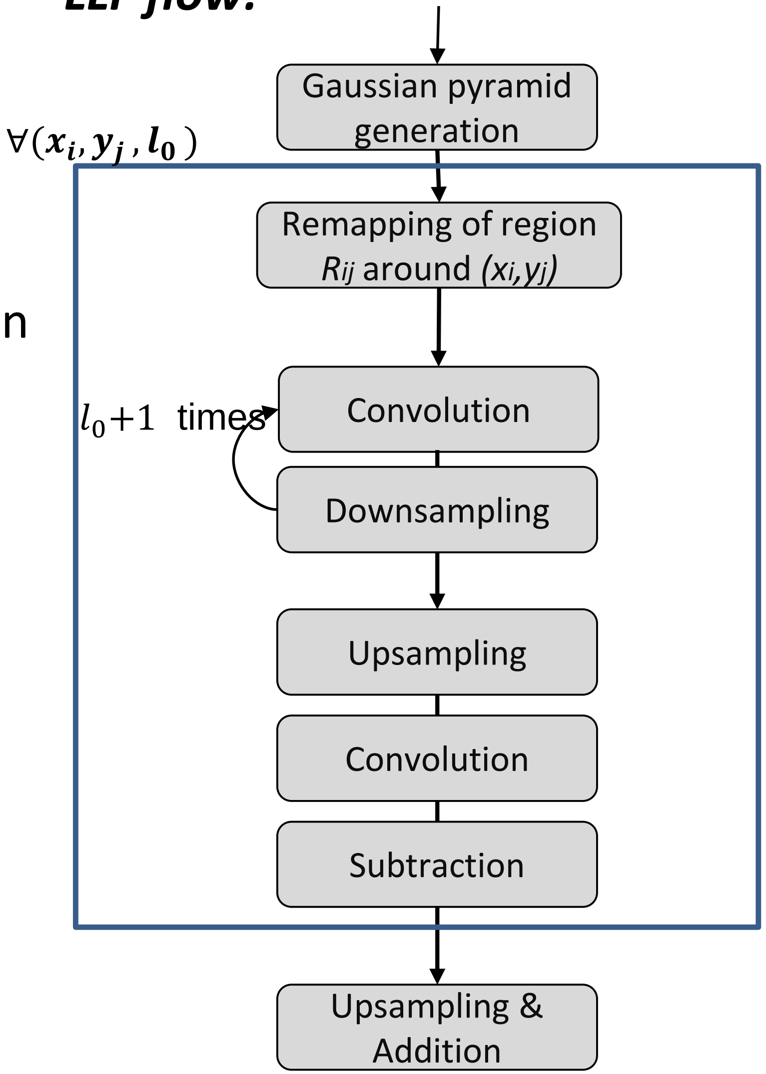
LLF: 1 command buffer invocation for each pyramid level

VO: up to 20 command buffer invocations per frame

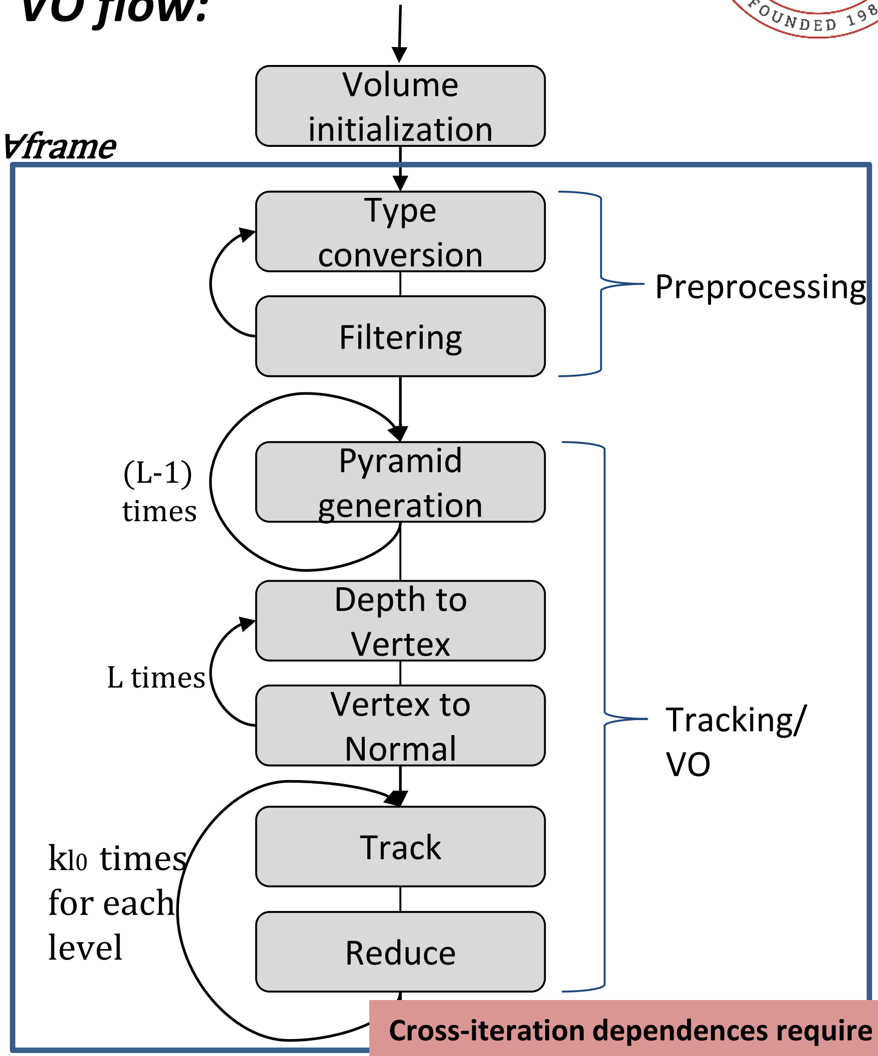
At the end of each frame, the flow transfers data to CPU for checking. Processing of next frame depends on the checking results.

Therefore large invocation overhead compared with optimized LLF.

LLF flow:



VO flow:



Cross-iteration dependences require that kernels are invoked from different command buffers

Discussion (II)

Compiler support for SPIR-V is still immature

- **spirv-opt** does **not** improve performance
- Many data types **not** supported by the SPIR-V IR
  - char, short
  - Array of structs in buffer elements

Discussion (III)

+ Programmer can leverage low level semantics of Vulkan (e.g. command buffer manipulation) to improve performance
  - Use a single command buffer and synchronize using memory barriers. Important for iterative applications.
  - Avoid frequent control and memory exchanges with CPU memory

– Additional complexity of Vulkan compute worth it?

– Vulkan compute is still work in progress!
  - Still cannot load an OpenCL kernel without intermediate compilation to GLSL compute shader.
  - SPIR-V Driver/compiler immaturity

Acknowledgements

«Acknowledgment: This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE
Project VipGPU: Very Low Power GPUs for Mobile Robotics and Virtual Reality applications (project code: T1EDK-01149)».

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Thank you for your attention
Questions?