Characterization of Energy and Performance Bottlenecks in Omnidirectional Camera Systems

by

Sridhar Gunnam

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2018 by the
Graduate Supervisory Committee:

Prof. Robert LiKamWa, Chair
Prof. Pavan Turaga
Prof. Suren Jayasuriya

ARIZONA STATE UNIVERSITY

July 2018

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

[Condense the below paragraphs]

With the advent of AR/VR technologies there is increasing demand to create content specific for these technologies which can either be virtual or real. Virtual content is created using game engines whereas the real world content need to be captured from cameras and then processed to the format in which the content can be viewed in AR/VR. The main characteristics of this content is to provide immersive seamless experience where user can view in any direction as fell as if they were teleported to that location. In order to have such immersive experiences, we need to bridge the gaps in several domains including optics, graphics, audio and video, etc. But during this project we focus on capture systems for 360 degree video.

What it means to have capture visually immersive real world scenes? Several researchers [Edvardo, AMD, Nvidia] predict that we need very high resolution(¿16k) and framerates(120+) to make the experience indistinguishable visually. Current 360 stereo video systems are used mainly designed for professional videography. They consist of several camera(18 in google's jump VR) which are bulky, and capture lot of data which will later be offloaded and used to generate AR/VR videos thereby limiting their usability. But in order to easily capture and share such experiences we need also need to focus on usability and portability of the devices to make 360 video mainstream in AR/VR.

We increase the usability and portability of the 360 devices if we can capture and stitch the panorama on the same devices.

Most of the software use traditional algorithms fit for offloading based approaches and doesn't consider power budget for implementing 360 capture using a low power portable device. 360 video is essential for VR, but capturing and stitching them in real-time is limited by battery life. Even if battery technologies improve, capturing and stitching 360 video will have heating issues, thereby increasing skin temperature. In-order to tackle the challenge of capturing and stitching on same device, we study the system level bottlenecks in energy and performance by building a prototype. We characterize the system level bottlenecks in terms of energy consumption and latency. [[[condense this]Our findings suggest that the main reason for the inefficiency is caused by building the system from off the shelf camera and traditional stitching algorithms. Conventional 360 degree is captured using a multi-camera rig and the expensive stitching is offloaded to powerful machines. Although some systems exist where stitching is done online, they are limited by output resolution, framerate and battery life. We show that the inefficiencies in the pipeline due to lack of hardware algorithm co-design. In this paper we study the data flow of the stitching pipeline by building a protype using 6 camera system. We analyse the energy and performance bottlenecks in the pipeline and analytically evaluate the mechanisms like using motion vectors to reduce temporal data access and computation, use raster buffers instead of full image to optimize on memory, and chip area.

Although commercial 360 degree solutions exist, they are mostly used for capture and stitching is offloaded to powerful machines. This limits the usability of 360 in VR and also portability and for heating. Our goal is therefore to build a 360 camera system that optimizes the entire pipeline both in hardware and software. We characterize the traditional pipeline and propose

Characterizing monoscopic, stereoscopic, bottlenecks, optimization]

The document is organized as follows, in chapter two we discuss about the back-

ground and related work, in chapter three describe about the general stitching pipeline for VR panorama generation, and the prototype system design. In chapter four we present the evaluation results of the prototype system design. We then discuss proposed optimizations in chapter five and our discussions and conclusions in the in chapter six.

The contributions of our work are as follows:

1) Build end-to-end system for capturing 360 degree video.

2) Characterize Individual Stage energy and latency and highlight the bottlenecks in the system.

3) Propose architectures to optimize end-to-end data flow and data abstractions needed at sub-system level. i.e optimizing the spatio-temporal redundancies in the data and computation.

Chapter 2

BACKGROUND

In this chapter, we will discuss about different forms 360 videos, then about the systems used for capturing each of these different forms. We then provide some necessary background to understand the image stitching pipeline.

2.1 Different types of 360 Videos

There are mainly ways 4 types of 360 capturing viz, monoscopic, Omni-directional Stereo(ODS), and Light Field cameras. In this work we will focus on monoscopic and stereoscopic cameras.

(a) Pair of spherical fisheye images (b) Equirectangular projected output

Figure 2.1: A 360 degree capture and corresponding panorama

2.1.1 Monoscopic 360 Degree

Fisheye lenses allow image sensors to capture images within an ultra-wide hemispheric field of view. With two fisheye-lensed sensors that capture complementary fields of view each of over 180°, the pair of captured images can be processed to

achieve over a spherical 360°x 180°area, as shown in Figure x. The equirectangular projection format is a common format for 360°x 180°images, allowing remapping to other projections for convenient viewing.

2.1.2 Omni-directional Stereo(ODS)

ODS output consist of two panoramas one for each eye, and provide the binocular stereo needed for perceiving depth information of the scene with respect to the view point of capturing device. In order to generate such output, we need to capture stereo information from all the viewing directions. Instead of capturing from all the viewing directions, we capture in certain directions, equally distributed over the 360 degree viewing angle and later process them to get the virtual camera viewpoints. We finally get the two panoramas one for each eye, which helps see the 360 view with depth.

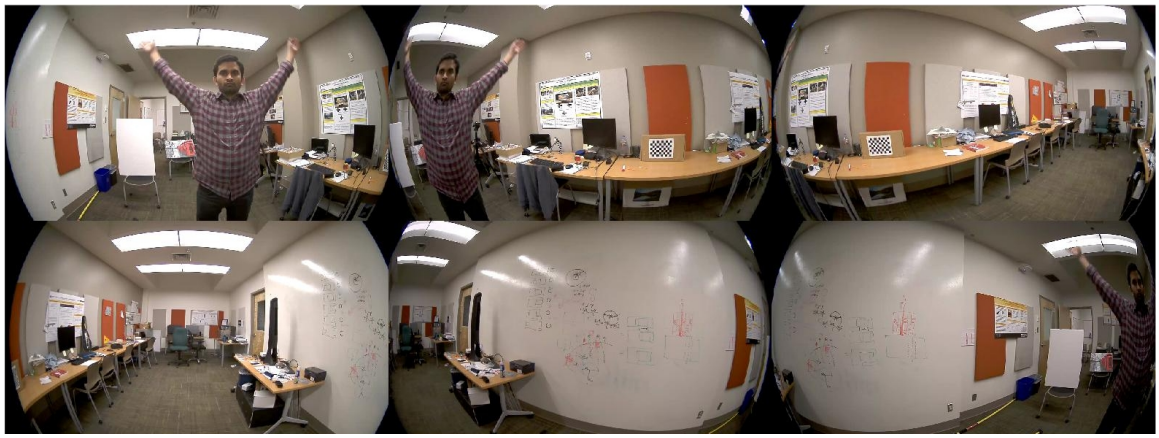


Figure 2.2: Six camera views covering full 360 Hfov

2.2 Key Words

VR Panorama related: FOV: Binocular Stereo: Frame rate: Resolution:

Camera Related: Pinhole Camera: Fisheye: Equirectangular Projection: Camera Calibration Process: Intrinsics: Extrinsics:

Figure 2.3: ODS panorama, one for each eyes.

Stitching Algorithm Related: Optical Flow: Temporal Regularization: Pyramid: Video Compression:

Compression Related: Video Compression Terminology: Frame Types: JPEG Compression: Motion Blocks: Motion Vectors:

Image Representations: RGB Vs HSV Vs YCbCr(YUV) RGB is mainly for display's. HSV is easier as we have intensity channel is available. For vision applications intensity channel is very useful, for graphics people it's easier as they can change the Hue - color, Saturation - colorfulness(intensity of color, shades of color), value - overall intensity intensity of image.

Y'CbCr is used to mainly for storage and transmission efficiency. Y' - Luma, which is intensity of gamma corrected RGB image. (We use gamma correction, which is non linear encoding of luminance/brightness to adjust for the way humans perceive light and color, i.e we perceive the variations in darkness more than the variations in brightness. And also we perceive light more than color.)

The more perceivable luma (Y') is used with high precision, whereas color(Cb Cr) can be of low precision, thereby reducing the size of the image.

2.3 Related Work:

Many have argued [Edvardo Hotmobile, Nvidia, AMD] that we need resolutions greater than 16k and frame-rate greater of 240 for true immersion in VR. At such higher framerate and resolutions there is lot of information that is redundantly captured, processed and transferred. Therefore, in our work, we study how the energy and latency of each stage get affected by the output resolution and the characterize the bottlenecks in greater detail.

Existing systems

Many existing systems like Google Jump, Facebook Surround capture and compute on enormous amount of data consuming several hundreds of watts of power to stitch images in realtime 30fps. Google Jump, Facebook Surround, Mega Stereo, Samsung Gear 360

Differentiating our work

Bottlenecks in the existing systems

Compression Related Work: Mobisys [Rubrix] Conduit encoding: https://avp.github.io/conduit/ In this project, they are using region based compression and other smart techniques. Although these works are for VR streaming, they share common techniques like reducing bandwidth by managing the image representation and decreasing latency by close integration of stages in VR streaming. Pyramid encoding: https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding-techniques-for-360-video-and-vr/ I think facebook's pyramid encoding seems a good idea for streaming, at that scale of facebook's needs. But I have some criticism for cube projection, as they don't use region based compression. The pyramid encoding is a good technique though, to reduce image file size

Chapter 3

DATA FLOW

[todo] Block Diagram - With different stages. - And Explain Data Inputs and Data Outputs of each stage.

The end to end system consist of four main stages viz., image sensor, image signal processor(ISP), processor, and Off-chip memory. The image sensor captures raw images, which are processed by ISP to generate RGB images, which are used by processor/accelerator, and the DRAM supports for storage of images and data for all the above stages. The ISP is typically integrated with the processor SoC, and Camera and DRAM are implemented in separate chips.

3.1 Monoscopic 360

Fisheye lenses allow image sensors to capture images within an ultra-wide hemispheric field of view. With two fisheye-lensed sensors that capture complementary fields of view each of over 180 °, the pair of captured images can be processed to achieve over a spherical 360 °x 180 °area, as shown in Figure 2. The equirectangular projection format is a common format for 360 °x180 °images, allowing remapping to other projections for convenient viewing. To create equirectangular images, the paired fisheye capture data undergoes multiple stages: 1) Projection Mapping: The equirectangular image is populated by sourcing image pixels from the fisheye images along a (spherical coordinate âĘŠ polar coordinate) projection map. As projected pixel coordinates typically fall between integer pixel coordinates, the algorithm typically either pulls a nearest-neighbor pixel or a bilinear combination of a neighborhood of pixels. 2) Correspondence: As the two fisheye cameras do not precisely occupy the

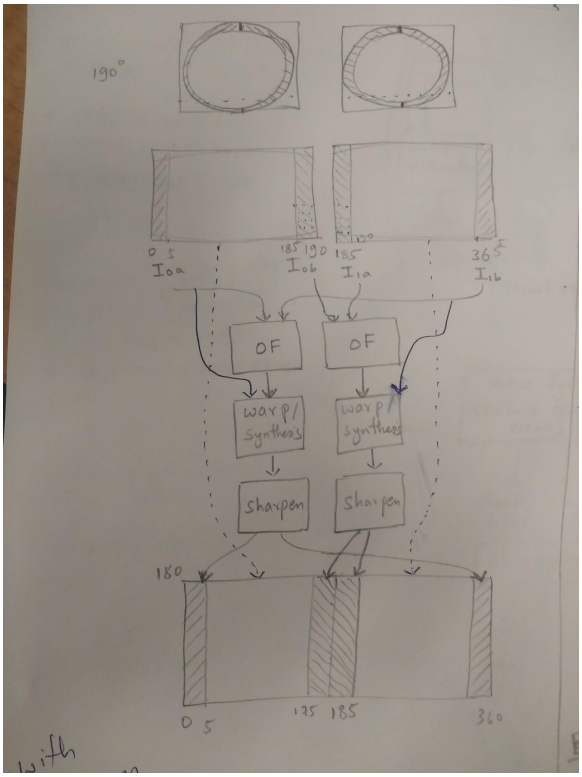


Figure 3.1: Data-flow block diagram dipicting different stages and their inputs and outputs in the end-to-end pipeline in monoscopic 360

same point in space, objects at the edges of fisheye images appear in different positions in the images, dependent on their distances from the camera. This phenomenon is called the parallax effect. To ensure that objects appear properly, a correspondence algorithm identifies matching visual features across image pairs, warping the projection to reduce object seams in the image. 3) Blending: Even after projection and correspondence suggest image overlay coordinates, intensity variations from misalignments still occur between the two projected images at the stitching boundary. The blending stage combines the images through a weighted sum of pixel values to generate a seamless 360 degree image with a smooth transition. 4) Compression: To reduce the bandwidth at the capture, networking, or storage interface, images can be compressed into representations that use smaller file sizes. Lossy compres-

sion schemes, e.g., JPEG/MPEG, allow dramatic reductions in file size by discarding information that is considered to be perceptibly irrelevant.

Hardware

Hardware: Dual Fisheye Camera

Monoscopic Stages: The stages may include Intensity compensation, fisheye unwarping, alignment, blending.

Figure 3.2: Dual fisheye gear 360 device used for capturing monoscopic video

3.2 Omni-directional Stereo System

The overlapping left and right images.

Hardware:Six Camera Rig. For our prototype camera design we use six IMX-274 cameras for capture and Nvidia Jetson TX2 for stitching. The Camera and Jetson specs are shown in figure 3.2.

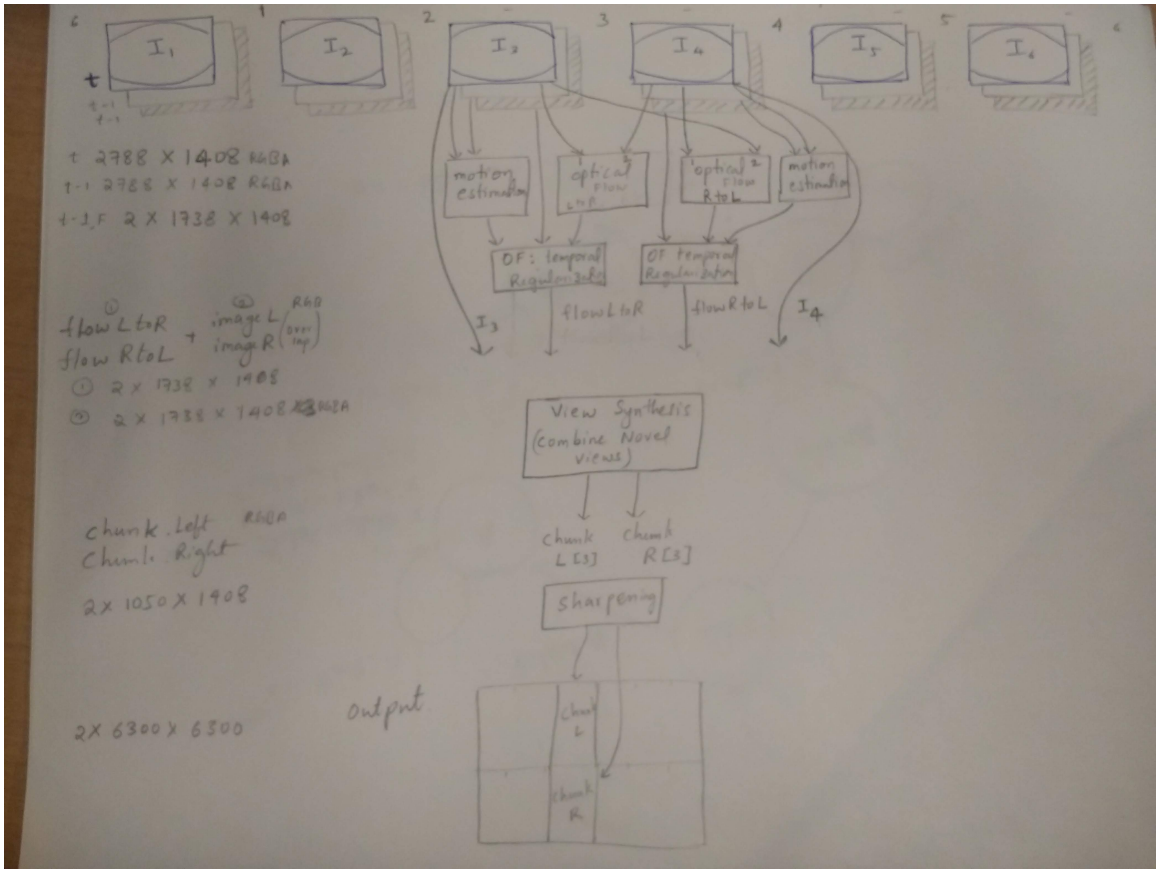


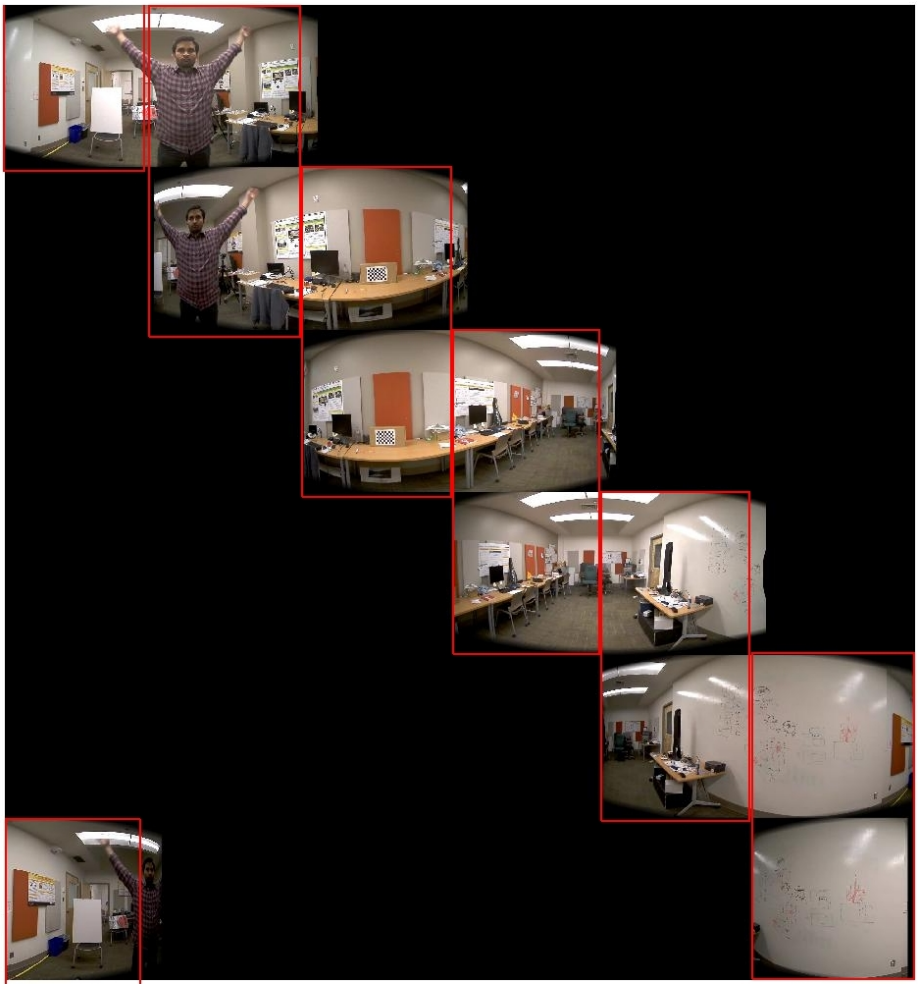
Figure 3.3: Data-flow block diagram dipicting different stages and their inputs and outputs in the end-to-end pipeline in ODS

Camera Name	Sony IMx274
Output Image Size	Diagonal 7.20 mm (Type 1 / 2.5) aspect ratio 16:9
Number of Effective Pixels	3864 (H) x 2202 (V) approx. 8.51M pixels
Unit cell size	1.62 um (H) x 1.62 um (V)

Software: Nvidia libargus Camera API, openCV, C++

Data Flow Diagram - With different stages

The main goal of this work is to reduce bandwidth requirement and remove redundant computation during different stages.



v3.jpg

Figure 3.4: Equirectangular Projection of first and second camera frames

3.3 Challenges with existing systems

3.3.1 Energy

Data and power numbers for Google Jump VR: Number of cameras: 17 Data Generated per each frame by 17 cameras(in bayer): 816 MB

Data bandwidth requirement in Gb/s : 47.8 Gb/s (@30fps, compressed data(1:8 compression ratio))

Power: Camera Capture and basic processing at camera: 50 W DRAM power(approx.)

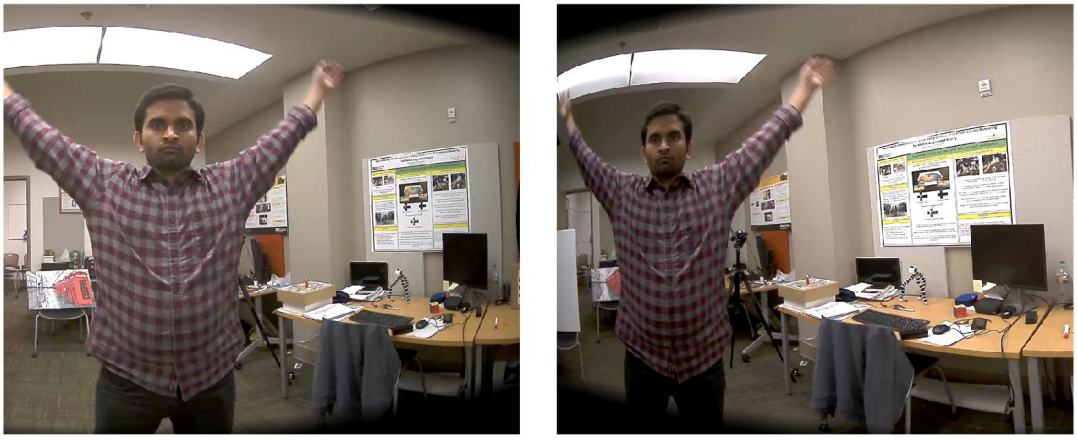


Figure 3.5: Optical flow inputs: Overlapping regions of adjacent camera images. Equirectangular Projection of first and second camera frames

: 61 W (30 W corresponds to store data generated by 30 frames of all 17 cameras) On chip LVDS power(approx): 19 W (For transferring one frame of all 17 cameras) Compute Power: (very high, done offline using multiple GPU's) Facebook also has 360 stereo solutions, one with 24 cameras and other with 6 cameras. I may need to find data for them as well. Consumes similar amount of power.

3.3.2 Latency

Several hours for final ODS generation.

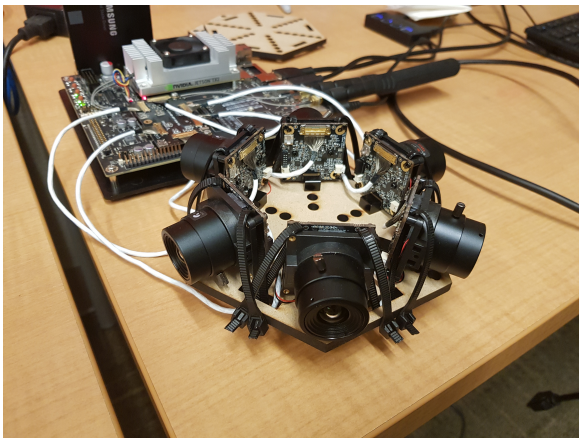


Figure 3.6: ODS Camera Rig

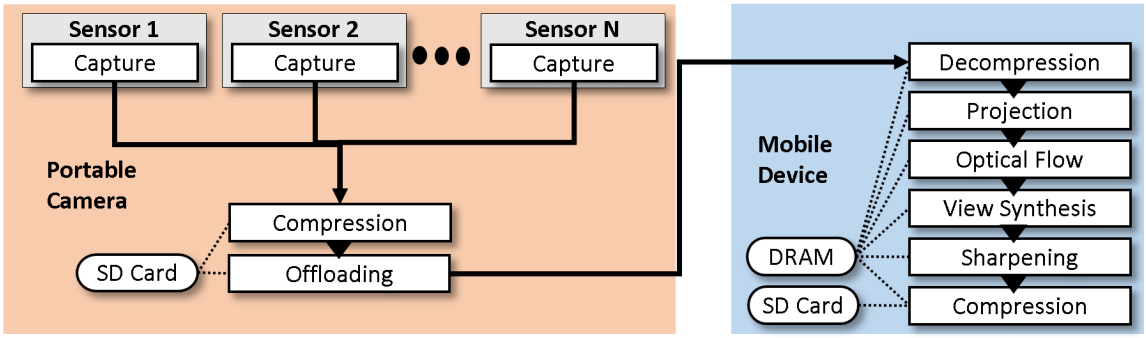


Figure 3.7: General Off-loading based system pipeline

Chapter 4

CHARACTERIZATION

The goal of the work is to characterize the energy and latency of end-to-end Omni-directional(OD) Camera systems both in the hardware and software pipeline and propose optimizations. As the existing OD camera systems are built from off the shelf camera devices and uses the conventional stitching algorithms, they capture, transfer redundant data and perform redundant computations. The redundancy can be attributed to the sptio-temporal correlation between the frames. The main challenge in OD panorama generation is to understand the data flow across the system and to make decisions on data abstractions needed at different subcomponents to reduce the total system power.

We characterize both monoscopic and ODS camera systems. The difference between them is the number of novel views needed is significantly higher for ODS compared to monoscopic. Both the systems at core use optical flow based stitching, so we will characterize generalized flow based stitching system and notify any important differences between monoscopic and ODS when necessary.

4.1 Measurement Methodology

Jetson has INA3221 monitors and I2C capabilities to read voltage, current and power for different rails on the SOC and IO. The can also moniter the CPU, GPU, memory clock frequencies. For evaluation we measure the absolute energy of the system and the difference between the idle and active state for individual stages of the pipeline. We also use nvidia tegra stats command to check the clock frequencies

of different components like CPU, GPU, Memory Controller for validation.

The latency of the camera capture and ISP is defined by the framerate(i.e throughput), where for computation stages we measure the latencies in terms of CPU runtime of individual software components in the stitching pipeline.

One of the critical components of stitching pipeline is optical flow which can take several seconds to compute each output frame on low power embedded CPU. Therefore for realistic estimation of optical flow for accelerator based design, we measure the power and latency of optical flow implementation on zynq FPGA board.

4.2 Energy Characterization

Individual stage energy

Power Rail	Diff. Current(mA)	Voltage(mV)	Energy (mJ/frame)
Sony IMX-274 (Camera)	375.4	3336	41.7
ISP+CODEC (TX2)	102.7	19152	65.6
ARM-A57(Capture + Stitching)	16.4 + 120 (16s)	19144	10.5 + 2296*16
DRAM (Capture + Stitching)	260.4 + 105 (16s)	4792	41.6 + 105*16

Dense optical flow at resolution 1280 x 720 ; 170 frames/second; Power = 4.8 Watt; Frames/Watt = 35.4

Note: For the calculating the energy for frame in the above table, the camera capture is configured to 1920x1080 resolution at 30 fps, and the output resolution is 3k. [make absolute energies instead of diff. i.e active -idle]

As seen in the above table, the stitching in software is highly expensive for CPU,and DRAM blocks, i.e the computation stage energy dominates the capture stage. CPU runtime to render each output frame of 3k is 16 sec, which increases

the both the energy and end-to-end latency. The other options include using GPU's, FPGA, ASIC's. From Nvi Therefore, we approximate the energy and latency for FPGA based accelerator based on Xilinx's implementation of optical flow on Zynq board, discussed in chapter5.

4.3 Latency Characterization

4.3.1 Individual Stage latency

We define stage latency and end-to-end latency for clarity. End-to-end latency is the cumulative latency of all the individual stages. The stage latency is latency of individual stage to process a single frame.

For camera system and ISP stages the stage latency is derived from through-put, i.e inverse of fps. The end to end latency is as given in NVIDIA camera API documentation, which is one frame latency for camera stage, and one for ISP stage.

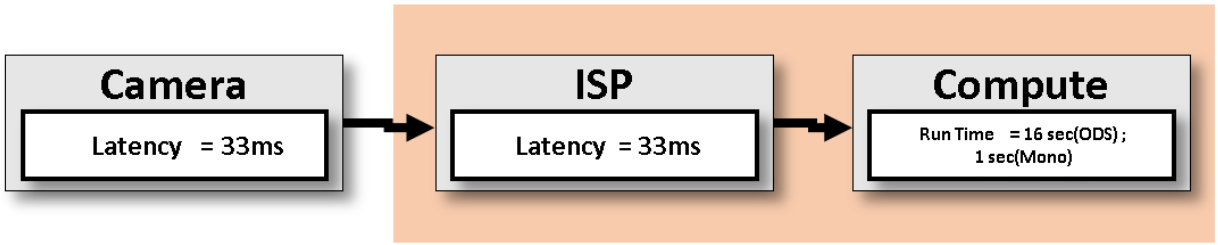


Figure 4.1: Latency of Individual Stage

We measure the latency of computation in terms of CPU runtime. The optical flow, view synthesis and image sharpening stages take 98% of total CPU runtime. Of this the major component is optical flow generation which consumes [] percent of runtime. The sharpening stage doesn't seem to benefit the image quality, so we

discard the sharpening stage entirely in the current pipeline and leave it for future work. We focus on the optical flow stage which dominates both in terms of memory usage, and computation.

4.3.2 Optical flow runtime breakdown

Time for each Pyramid search(98%) as shown in fig 3.1.

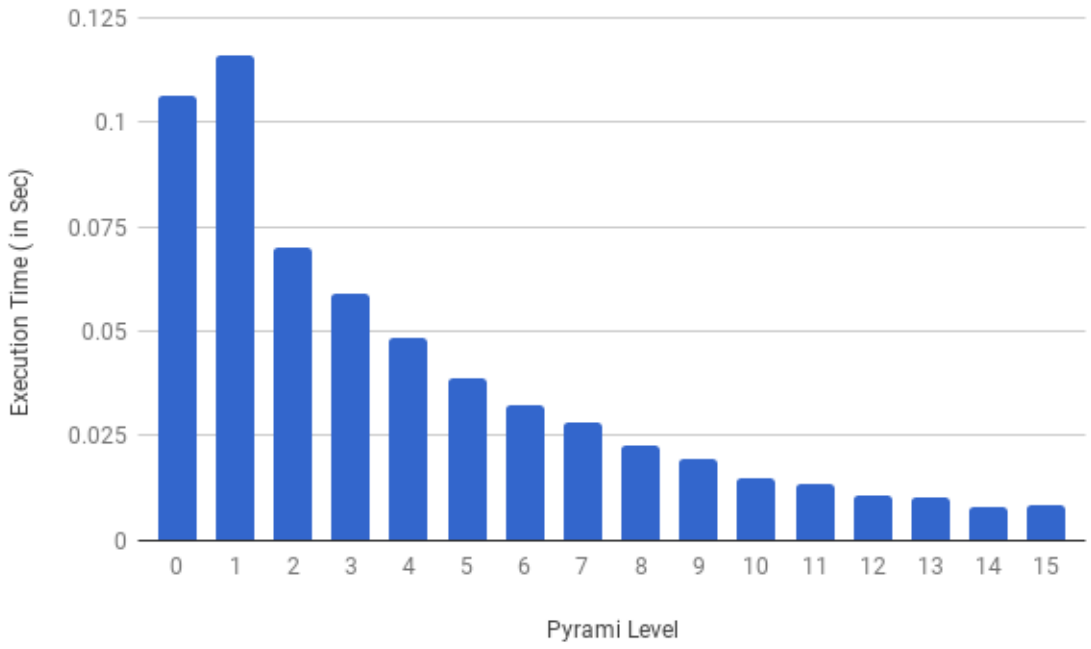


Figure 4.2: X-axis shows the pyramid level and Y-axis shows the runtime for OF

4.4 Camera Sensor Configuration

Sensor Response Time

As the ODS consist of several cameras, it makes sense to reconfigure or turn cameras or off based on the application needs and the scene dynamics. If the camera is not moving and certain portions of the camera views are static, the cameras can be

reconfigured dynamically to reduce framerate, resolution, or even turn them on and off as per the needs. We observed that the reconfiguration latency is one frame delay if there are no outstanding requests, and if there are pending camera requests, they will be served first before requesting the frame with new configuration.

The optical flow works well when the image is has high dynamic range. It is possible that some of the regions in the 360 degree view can be in low lightning, while other are in good lightning conditions. In such cases the stitching fails and can have severe artifacts. We can improve the dynamic range of the particular cameras in low lightning by reconfiguring the camera exposure time dynamically. But such approaches doesn't consider the end to end latency of the cameras and camera movements. To account for camera movements, IMU sensor data can be used to make the camera configuration decisions independent of CPU to accelerate the reconfiguration tasks.

The figures show the differences in low light capture with and without brightness correction. [Integrate the images with histograms]

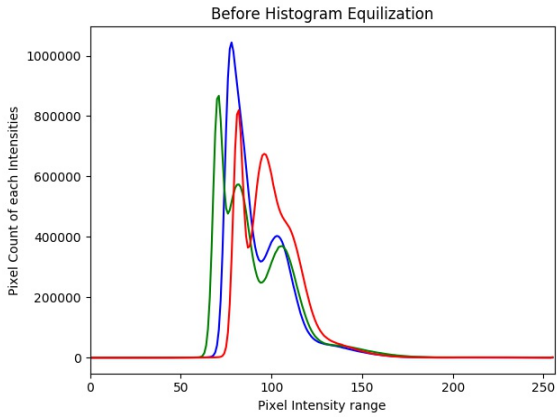


Figure 4.3: X-axis shows the pyramid level and Y-axis the runtime tile search and propagate.

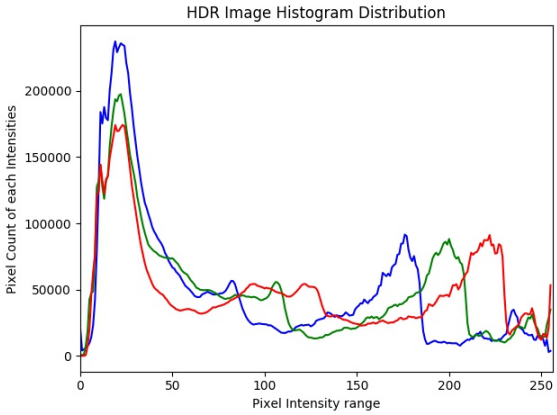


Figure 4.4: X-axis shows the pyramid level and Y-axis the runtime tile search and propagate.

4.5 Design Scalability

Runtime scalability with the resolution

As discussed in chapter 2 related work, the resolution required for next generation VR is at-least 16k and frame-rates greater than 120. For our evaluation, we assume that energy scales linear with frame-rate and focus our evaluation on scalability of increasing resolution. We can see in fig that even the runtime scales almost linearly with the resolution. Notice that optical flow dominates the total runtime, followed by view synthesis and image sharpening. We also measure the frequencies of CPU, DRAM controller with increasing resolution and observe [linear] dependency of clock frequency on the resolution.

The main takeaway is that the existing hardware and software scale linearly with the increasing resolution and framerate, which is bad considering the VR panorama requirements. We therefore propose directions to exploit the spatio-temporal redundancies within the frame and across the frames to reduce the data flow and computation. We propose rasterbuffer based designs to decrease the chip resources,and using

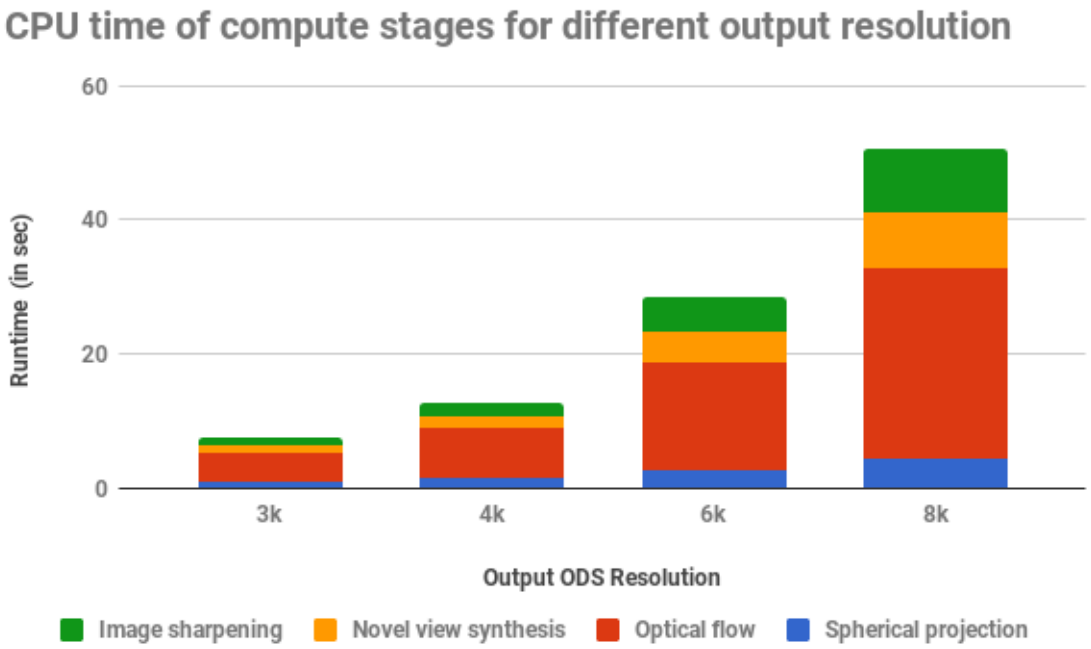


Figure 4.5: CPU execution time of different compute stages. X axis has different sub-stages in optical flow and Y axis correspond to energy per frame.

data abstractions at different hardware IP blocks to share data to reduce temporal redundancies(eg. motion vectors from ISP can be used by optical flow stage, thereby removing the necessity to store previous frames and recomputation of motion data at a later time. The same motion vectors can be used to encode spatial frame data to reduce redundant data transfer).

Resource scalability with the resolution

We measure the DRAM capacity required and bandwidth needs as we increase the resolution as a parameter for resource scalability. Higher capacity indicates the need for better encoding schemes and high bandwidth can indicate the temporal redundancy in the data, thereby increasing the bandwidth requirement. For 3k, 4k, 6k and 8k output resolution.

Although we built a system where all the cameras are capturing at same resolution and framerate at a given time, we expect the future cameras make these decisions dynamically to save power. Therefore, we measure the efficiency of capture and ISP processing at different modes of operation and measure the efficiency of capture and processing in power consumed per pixel at different modes.

DRAM memory size

Stage	3k	4k	6k	8k
camera Input	-	-	-	-
ISP	-	-	-	-
Motion Estimation	-	-	-	-
fish2EqRect Projection	-	-	-	-
Optical Flow	-	-	-	-
Sharpen	-	-	-	-

o/p Resolution	3k	4k	6k	8k
Avg. DRAM BW	-	-	-	-

4.6 Evaluating data-flow redundancies

Evaluating redundant computations in optical flow. Accuracy Vs Energy&Perf tradeoff. Next we evaluate the optical flow for a scene where foreground has movement and background is static. The temporal flow difference is found out to see the variation of flow. As expected we see the flow change only for the regions where the objects are moving. This observation suggests that accuracy of optical flow can be trade-off

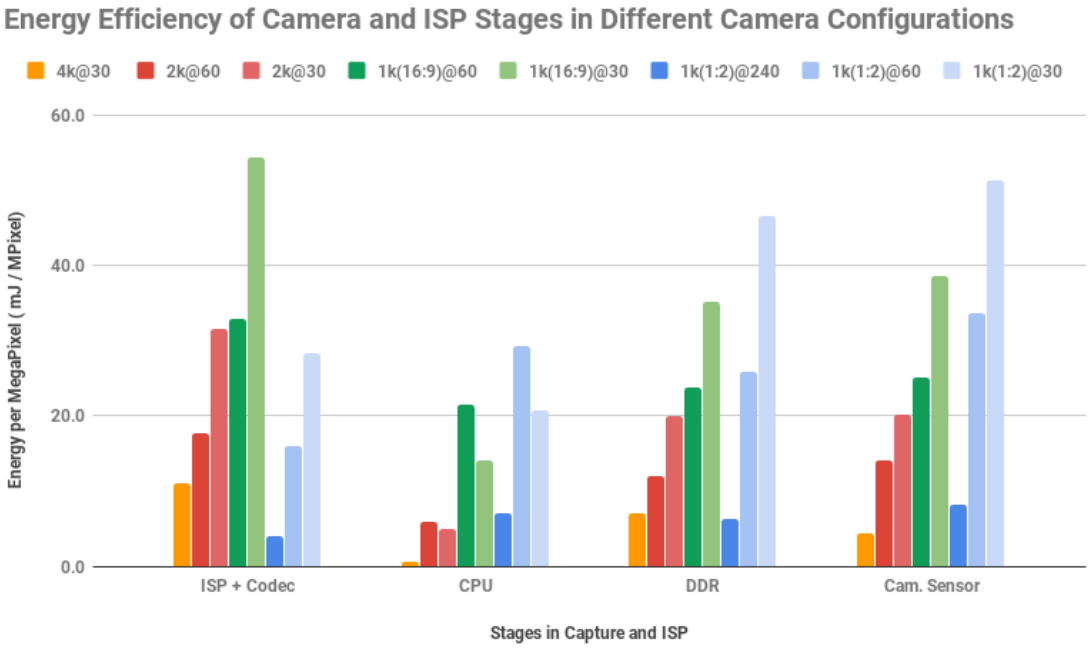


Figure 4.6: Power Efficiency of Camera ISP Stages in different configurations

with computation time to save energy and latency. It also shows that the accuracy drop is less than _ percent which can be acceptable

4.7 Misc

Increased frame-rate

What is the percentage of new data ffmpeg I-frame size to the P-frame ratio. we usually have I,P, and B frames in a compressed video, but in case of realtime compression we will have only I and P frames and disable B frames as it adds latency to the pipeline. Typically I-frame to P-frame is about 4 times and one I-frame occur for 30 P-frames. This implies we save a lot on interface power if we can push the computation to near sensor. [Numbers - savings from IO datarate reduction]

3) Survey of Camera and ISP stage energy breakdown Camera Sensor and ISP power directly taken from literature. Computation Power split into sub stages.

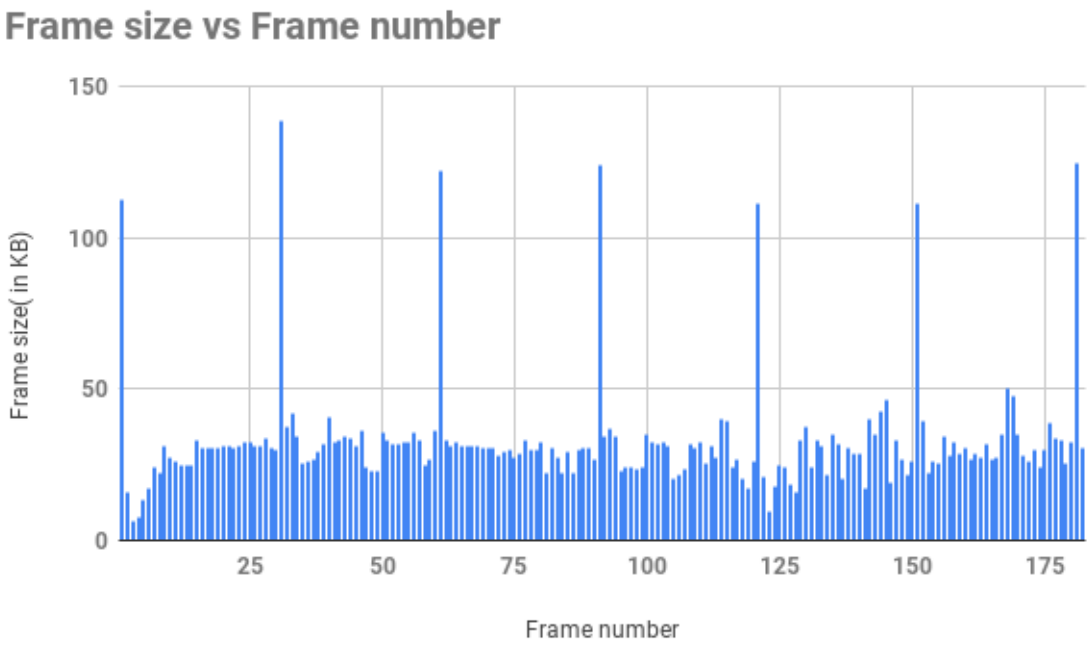


Figure 4.7: Framesize of I and P frames

For power characterization of camera sensor and ISP, we run the camera in different resolutions and framerates and see how the various sub-component power changes. The components include Camera Sensor, I/O, ISP, CODEC, DDR, and CPU. The ISP, and CODEC power are combined as they belong to same SOC voltage rail.

The most used configuration for our project when all the six cameras are capturing 1920x1080 @ 30 fps. At this configuration below is the split of different component power.

e) Quality Tradeoff's with input resolution

Sharpening Reduction in fidelity of unwarped image, as interpolation is not being done. Reducing number of pyramid's f) High motion Vs low motion differences. Size of motion vector to that of size of full frame. g) File IO power h) Breakdown in terms of type of Memory used i) Breakdown in terns of type of Computation j) Breakdown

in terms of IO bandwidth bottlenecks

Chapter 5

PROPOSED MECHANISMS

Technique ⇒ Benefits

1) Hardware software co-design: Re-using motion vectors generated by ISP stage to reduce the re-reference of previous image frames to calculate motion vectors for optical flow generation.

(How to check if we need to compute everything with intensive vision algorithms or just use the precious results, what granularity to compute. Eg. Pyramids updating. Reactive to reconfiguration and powering on and off.)

a) Reduction of DRAM capacity requirement

b) Reduction of DRAM bandwidth requirement

c) Reducing end-to-end latency in generating dense optical flow

The calculation of optical flow is the major bottleneck in terms of both memory usage and computation. The inputs for the optical flow include the current and previous camera frames of two adjacent cameras, their alpha channel and the previous optical flow. The need for temporal frames is to detect motion that is used of temporal flow regularization. Since the main memory is limited, the previous frames are usually stored in disk which increase the flow computation latency, and increase the DRAM capacity requirement and the bandwidth requirement. Instead of saving the temporal frames for motion estimation, we can leverage the motion estimation hardware IP's that use optimal DRAM and completely remove the need to go through the disk. We model this by precomputing the motion vector and feeding with the image data. From our emulation, we found reusing the motion estimation reduces the disk utilization

by __ %, the DRAM energy by __ %, improves the end-to-end latency by __ %.

High Level Synthesis: HLS provide constructs to generate hardware that uses parallelism, pipelining, data reuse and streaming operations. On top of these we also need to do bitwidth optimization.

2) Data driven execution: Use of motion vectors and previous optical flow to update the pyramids to make use intrinsic properties of foreground, background and motion in the scene. a) Reduces the number of computations i) For building pyramids. ii) For calculating SAD(Sum of Absolute differences) during pyramid block matching. b) Reduces off-chip accesses c) Reduce end-to-end latency

Adaptive stitching pipeline * Not all regions of the image have same level of difficulty for stitching. In an outdoor filming, most of the top portions are covered by sky, and ground is covered by road. So if those regions can be stitched with lesser effort(i.e reducing the number of iterations in stitching). Since we are going to do flow based stitching, the number of iterations is number of tiles size reductions in the flow based iterative stitching.

3) Hardware Accelerator: Streaming Architecture: Using raster buffers across the entire optical flow pipeline.(Number of rows is constrained by maximum motion in the scene) a) Helpful for scalability to higher resolution b) Reduces size of local SRAM and off-chip memory access.

Power Rail	Diff. Current(mA)	Voltage(mV)	Energy (mJ/frame)
ISP+CODEC	102.7	19152	65.6
CPU	16.4	19144	10.5
DDR	260.4	4792	41.6
Camera	375.4	3336	41.7
CPU	[]	[]	[]
Accelerator[Zynq]	[]	[]	[]

Case for low power 360 capture. Real time Stitching 30fps ¿4k resolution with low power. Latency of GPU, CPU makes them unusable for vision tasks in AR, VR. Case for algorithm software Co-Design for a line buffer based streaming architecture.

GPU Based Acceleration. NVIDIA Tesla: A unified graphics and computing architecture. E. Lindholm, J.Nickolls, S. Oberman, and J. Montrym. IEEE HotChips 2008 The paper gives an introduction to graphics and parallel computing capabilities of Nvidia Tesla. The organization of GPU from hardware and software aspects is explained in detail. A GPU, in general, will have one or more streaming multi-processors(SMT). The SMT in Tesla consist of 8 Streaming Processor cores. The serial part of the code is implemented in CPU, whereas the Graphics and parallel computing are done in SP cores of GPU. The SM is hardware multi-threaded. It creates and executes a group of 32 threads which are of the same type of operation. They are called warps. GPU has three types of memories local global and shared. Local memory is allocated each thread and is physically located in DRAM. Shared memory is located near SMP, and used by cooperative thread arrays(thread blocks). The global memory is in DRAM and is used by sequential grids(set of blocks) to communicate large data sets. The paper also talks about CUDA programming model.

In CUDA the parallelizable kernels are implemented in terms of grids and blocks and given to GPU.

1) Energy Characterization of end-to-end pipeline

Camera, ISP, Computation

Split of energy in computation

2) Runtime Characterization

a) End-to-end pipeline

b) Split in computation execution

3) Performing motion estimation prior to computation stage

a) Savings in DRAM capacity, bandwidth(Normalized)

b) Savings in DRAM Bandwidth

c) Savings in overall energy

d) End-to-end latency reduction

4) Optimizing of computation in pyramids

a) The execution time split for creation of pyramid, finding optical flow of pyramid, refining/updating the pyramids, upscaling the pyramid.

98 percent is to generate optical flow(dense pixel correspondence). But only 20-30 percent actually needs to be recomputed.

Main optical flow method time is 0.560256 Total time for entire optical flow is 0.584954

5) Sense the environment in gray scale and perform color mapping later? How much are you saving?

6) Egocentric motion

7) RAM-less architecture RAM-less architecture is possible for panoramic stitching

as we plan to process data and instantly stream the processed row data from one stage to another, instead of performing computation on full image. We need to make the computation stages generic so that it can be extended to other applications with the use of same hardware resources. We also need to work on how to program the computing units and coordinate communication between computing stages. When we generalize streaming architectures we may create idle resources in computation stages for which fine grain power gating and clock gating techniques can be used to save power from the gates that are idle.

There can be streaming applications where the DRAM might be necessary, in such situations our architecture should be able to support the DRAM with ease. Therefore our architecture need to be flexible to include DRAM if required.

Poster data

Recent advances in image sensor technology and lens designs have led to the emergence of portable spherical panorama systems, such as the Samsung Gear 360, capturing information to process into 360 degree x 180 degree images and videos. However, real-time, energy-efficient generation of high-resolution spherical panoramas remains a substantial challenge, as standard computational architectures are incapable of efficiently processing large amounts of data. Because energy consumption generates heat, creating imaging artifacts, e.g., lens warping, spherical panorama systems are constrained by a tight energy budget. This has led commercial implementations to offloading-based designs, in which stitching is done on the smartphone, and not in real-time. These implementations are incapable of scaling to large resolutions, due to limited and energy-expensive network bandwidth. Our proposed research will create designs that efficiently scale to ultra-high resolutions into the future, based around streaming spherical panorama architectures that do not rely on DRAM, memory storage, or other expensive I/O during processing. We estimate that this will create a

sub-watt architecture to generate and transmit 4K 360 degree video under 1 watt on the portable spherical capture device.

We propose two key research goals: (i) Establish an efficient RAM-less streaming architecture to direct fisheye sensor data into equirectangular output; and (ii) Study the effects of early in-sensor compression to reduce the transmission of data across sensor interfaces.

5.1 Stereaming feature detection and Correspondence

Conventional offload-based system architecture: Energy consumption creates a hard limit, or we will heat sensors and lenses, which degrades the quality of image capture. A conventional memory-based computational architecture on the capture device would consume prohibitively high energy consumption per frame, bottlenecked by memory operations on the many pixels of images with high spatiotemporal resolution. This has motivated commercial system implementations to leverage networked devices, e.g., smartphones for the Gear 360, to perform the processing stages, while the portable capture device is only responsible for capturing and transmitting lightly-compressed images. The offload-based system architecture is illustrated in Figure 1a. Unfortunately, the scalability of offload-based solutions is limited by the network interface. Networking for hiqh-quality panoramas is already prohibitively energy-expensive (500 mJ per 15MP frame), and slow (2 frames per second), requiring a networking power consumption of over 1 watt to achieve required bandwidth. An additional burden to the network interface and other system resources is that input fisheye images must be captured and transferred with high quality. Resolution and compression artifacts in the input image will greatly deteriorate the perceived quality

of the output image. Thus, image compression on the capture device is limited to low compression ratios, as fisheye capture data, further straining energy-efficiency. Finally, smartphone SoC architectures are ill-equipped for panoramic stitching, due to limited parallelism and a heavy dependence on memory loads and stores. This causes an overhead of roughly 4 seconds to generate a spherical panorama on a Galaxy S7. On a specialized device, such as the Gear 360, it is possible to incorporate application-specific hardware to enhance the device's features, e.g., Figure 1b. However, such hardware must be designed carefully; conventional architectures that rely on DRAM and storage consume substantial energy consumption, heating up the device, and draining the battery. Our proposed RAM-less streaming architecture, illustrated in Figure 1c, aims to perform the processing stages on the spherical capture device itself at sub-watt power efficiency. Generating the equirectangular image before transmission allows larger compression ratios for the transmitted output image while preserving perceived quality. Finally, the computational load on the networked smartphone is also reduced and high quality images are readily available for real-time streaming, i.e., without processing latency.

We propose an architecture for feature detection and correspondence that makes effective use of spatial locality towards pixel buffers and divide-and- conquer strategies, allowing an independence from random-access memory. We expect that features detection and correspondence operations can be derived from standard corner-based algorithms, but propose to study novel hardware designs for area-efficiency and energy-efficiency.

To further reduce the energy consumption of the system architecture proposed in Thrust 1, we target the image sensor physical interface as a bottleneck to energy efficiency. As temperature requirements force a substantial distance between

image sensors and processing units, sensor data transactions are notoriously energy-expensive. At full input resolution of 15 MP at 30 frames per second, using a typical Low-Voltage Differential Signaling (LVDS) physical interface consumes 2W to satisfy the >3 Gbps bandwidth, e.g., [8]. This is prohibitively high, and the Gear 360 typically is constrained to capture video at one half of this available resolution, while still consuming multiple watts of average power consumption. We explore the use of in-sensor compression to assist in an energy reduction, as compared in Table 1. Existing hardware solutions for JPEG and MPEG compression are plentiful and sufficient, dropping data

bitrates by substantial compression ratios (e.g., 8:1, 23:1, 46:1, etc.) and some image sensors integrate real-time JPEG encoders into their package [9]. As shown in Table 1, this can have dramatic savings on interface power consumption. Using this as our basis, Thrust 2 proposes system-oriented research for placement and utilization of compression hardware close to the sensor. Thrust 2 aims to approach research objectives of (i) processing on the compressed data, i.e., without decompression; and (ii) region-based compression quality.

5.2 High and Low Resolution combination

What are the regions of the image frame that are needed in high resolution in order to generate a high resolution output ODS video.

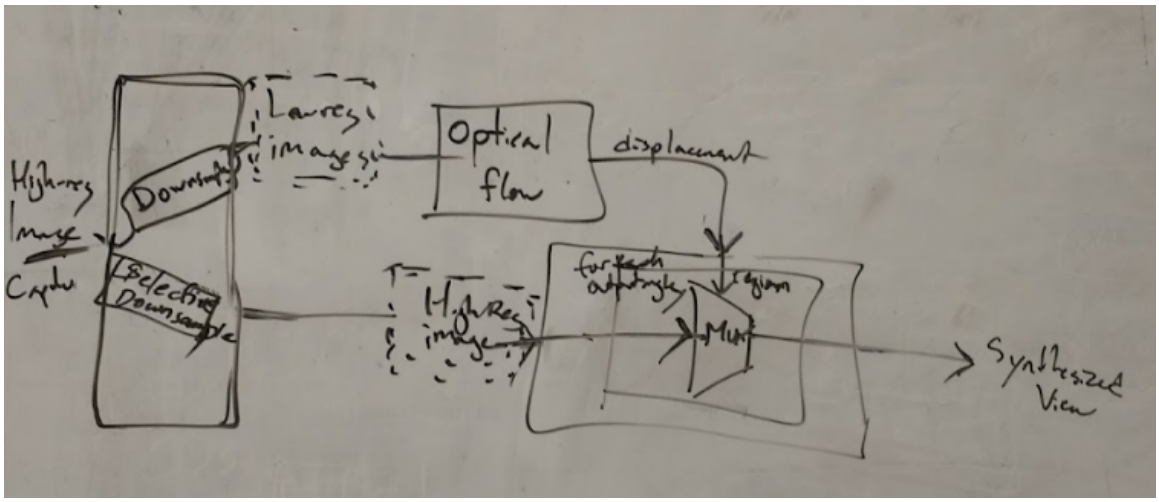


Figure 5.1: X-axis shows the pyramid level and Y-axis the runtime tile search and propagate.

Chapter 6

DISCUSSION AND CONSLUSION

Intro

6.1 Section1

Summarizing the proposed optimizations and future directions. DRAM-less Stacked Image Sensors ADC readout power, Rolling vs global shutter Abstractions for hardware software co-design a) How to find out which data to sense, send without the intervention of computationally intensive vision algorithms. b) How to detect them early in the vision pipeline c) Data Driven

———————————————— New Image Compression Techniques:âĂŃ

Instead of approximating 3D object movements as 2D motion block translation, we need to model the 3D model rotations and translations in compression algorithms.

Optical flow in spherical coordinates.

3) Develop methods to perform computation on compressed data.

4) On compression: Evaluate different compression algorithms and compression rates and how they affect data transfer and performing computation on compressed data. We may need to invent/improve existing compression algorithms according to requirements of our application. 5) Dynamically perform region based compression. If region based compression is performed dynamically, we need architect the system to enable communication between stages to provide necessary compression information. (This might be use case in continuous vision sensing for surveillance application where the region of interest could be a moving object. We may want different compression rates based on region of interest.) 6) In region based compression, the following stages

should be aware of which region is compressed and should accordinglyâĂŃ perform computation in it. We should check for feasibility to multiple compression ratios on the same image frame for different regions but still perform computations appropriately. ———————————————— Some of the research challenges include: Making use of as less cameras as possible. Computation Vs Capture tradeoffs Need for novel computation methods to reduce design size and data transfer. Can we generate High resolution Stereoscopic capture using Low resolution monoscopic 360 video capture + low resolution Stereoscopic capture. Image representation for high compression, decompression and view transformation. Can we generate one equirectangular representation from another with simple transformations. Where should this views be generated? Can we do some sort of encoding between the two views for highly correlated regions? (Assign ID to common regions, transmit only one of them) ———————— Communication and Power Regulators Humans communication is adaptive to situations. We speak fastly/slowly, change languages, our protocols are adaptive i.e we interrupt others sometimes/ and don't few other times. But the system interfaces are static and are not adaptive, they have fixed protocols and data rates. Can we get inspired from nature and make these interfaces more programmable and dynamically adapt to the requirements of the system?? What is stopping us?? Where do such adaptive interfaces comes into picture?? We do have some sort of dynamic nature at interface hubs. But are they programmable? ————————- Optimizing number of cameras using software modelling in Unity/xyz using multiple virtual cameras. Create virtual camera rig, set camera parameters of virtual camera and capture images [Stanford Work] Stitch images captured by virtual camera Compare with the ground truth, i.e the actual 3D environment https://blogs.unity3d.com/2018/01/26/stereo-360-image-and-video-capture/

6.1.1 future work

Future Work: Divide it into: Hardware/Software and New Technology

Sub categories of different domains and fields. Eg: Vision, Graphics, ML, Systems, Networking etc Graphics: Model generation Vision: Systems: Light Field Cameras

Challenges in immersive 360 degree capture for natural environments: Reflective challenges in 360 camera capture. Different cameras see different reflections, will the math still be the same. It won't be, because we have not considered illumination.

Filling the holes using AI

Image representations

Viewpoint aware static and dynamic scene recognition Integration of codecs Near sensor ADC Motivation for SAR or hybrid SAR to single slope ADC(state of the ART) Reducing computation

6.1.2 Research Questions

1) Data Flow: Reducing redundant computation and transfer.

Cause ⇒ Effect

Cause:

Temporal frame re-reference(reuse) for motion vector calculation.

Previous optical flow re-reference for temporal regularization of current optical flow.

Effect:

Increased DRAM memory consumption

Increased Memory Bandwidth

Increased end-to-end pipeline latency

Cause:

Re-computation of entire flow pyramids for each frame. (Pyramids of previous flow,

estimated flow on current frame, current gray image, and alpha channel, all in floating point)

Effect:

Increasing the number of computations needed for each frame.

(Differentiating between the background and foreground and motion vector inputs to reduce the computations. background information is available in the form of optical flow correspondence, motion vectors are available from the ISP/motion estimation block)

2) Data abstractions and formats? When and where should we find motion vectors? How much is size of local buffers at each stage? Make analytical producer consumer model? How much energy saving at ADC readout? How much savings because of DRAM organization? and bit quantization?

Data Abstraction and Data Representations

Equirectangular format stretches the region near poles thereby storing redundant data. It is also difficult to compress as the existing motion estimation for video compression[facebook] doesn't work on equirectangular format. Cubemap on the other hand reduces the storage of redundant information at the poles. It also helps in compression as we can use existing compression techniques. There is also a trend for equi-angular cubemaps which reduce the file size even further by mapping the spherical images onto smaller cubes.

Which is better format to represent 360 videos monoscopic videos? Equirectangular or cubemap or new formats which can represent spherical images that reduce both data storage and rendering effort? As these formats keep changing, we need

design systems that are independent of the type of projection format.

6.2 Conclusion

Characterization summary Combination of feature based and dense correspondence based optical flow.

REFERENCES

APPENDIX A

ODS STITCHING IMAGES

The fisheye camera images.
The Equirectangular projection.

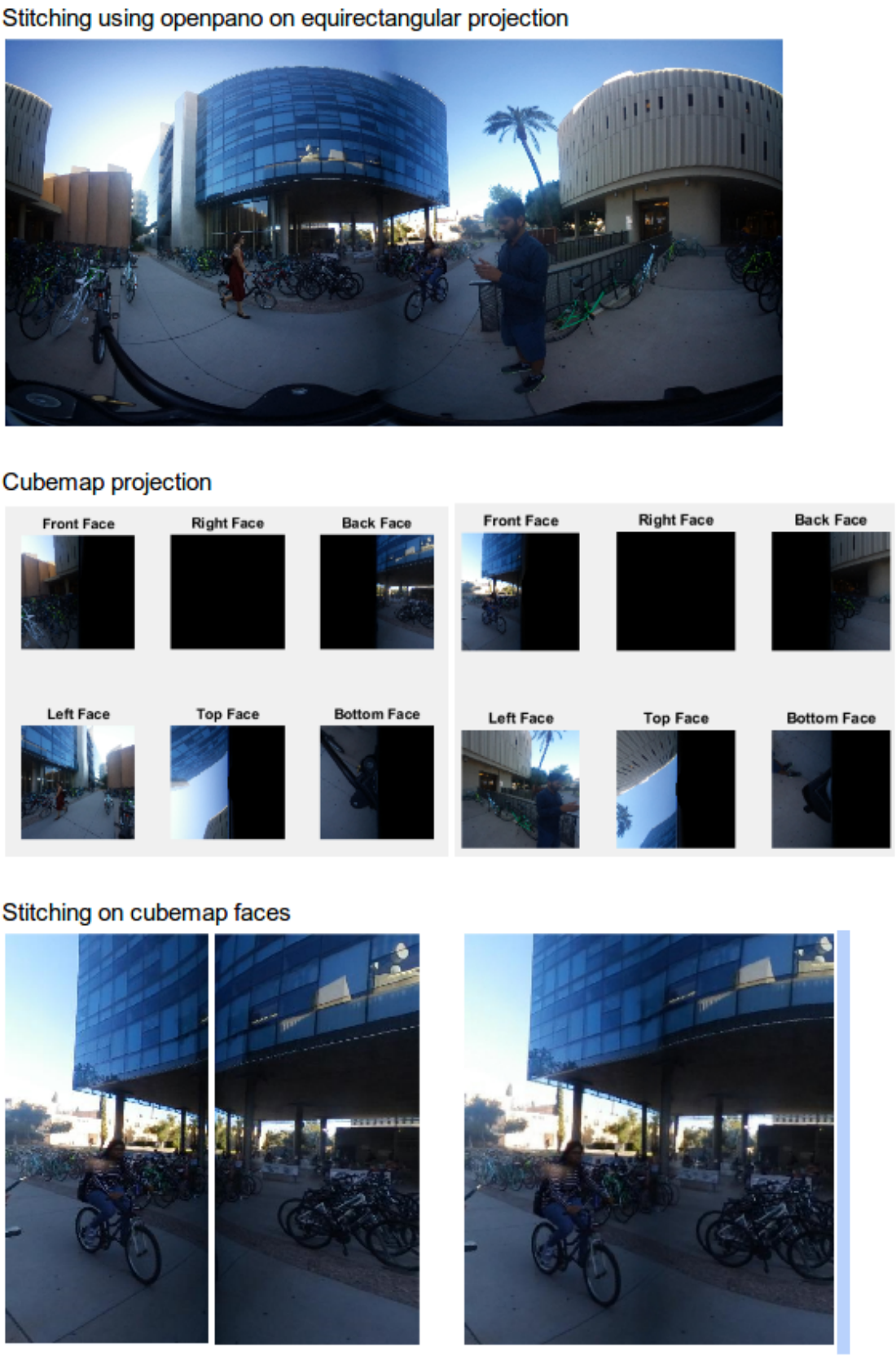


Figure A.1: Cubemap based Stitching, Path : /media/gunman/Data/fall-2017/research/mono360/equi2cubic

APPENDIX A

MAKING OF THE CAMERA RIG

Camera Calibration Process. Designing the supporting base.

Usecases

Capturing immersive real world scenes for experiencing in AR/VR. Why we need small form factor/ low power?

Mobile 360 capture, AR, VR, MR, Autonomous Driving.

Usecase1: 360 stereo capture for experiencing in VR headsets. [xx1] Sports live stream (only few powerful ones are sufficient) For general purpose capture and streaming, we need portable and long capture time capable. Existing VR cameras are available in small and large form factors but they have limited live streaming capabilities and don't have good battery life(60 minutes max).

Usecase2: In Mixed reality headsets(military training, gaming, etc) we capture and overlay virtual objects and display. So latency critical stitching.

Usecase3: Hybrid, 360 monoscopic and stereoscopic capture and display.