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

Page

LIST OF TABLES iv

LIST OF FIGURES v

CHAPTER

1 INTRODUCTION 1

2 BACKGROUND 4

3 SYSTEM OVERVIEW AND DATA FLOW 7

3.1 Prototype System Overview 7

3.2 Data Flow 7

4 CHARACTERIZATION 10

4.1 Measurement Methodology 11

4.2 Energy Characterization 11

4.2.1 Individual stage energy 11

4.3 Latency Characterization 12

4.3.1 Individual Stage latency 12

4.3.2 Optical flow runtime breakdown 12

4.4 Sensor Response Time 13

4.5 Design Scalability 13

4.6 Evaluating data-flow redundancies 17

4.7 Misc 18

5 PROPOSED MECHANISMS 20

5.1 Stereaming feature detection and Correspondence 25

5.2 High and Low Resolution combination 26

5.3 Data Abstraction and Data Representations 27

6 DISCUSSION AND CONCLUSION 28

CHAPTER Page

6.1 Section1 28

6.1.1 Section2-future work 30

7 RESEARCH QUESTIONS 31

8 CONCLUSION 33

REFERENCES 34

APPENDIX

A ODS STITCHING IMAGES 35

A MAKING OF THE CAMERA RIG 39

A SYSTEMS ARCHITECTURE SURVEY 40

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

3.1 X-axis shows the pyramid level and Y-axis the runtime tile search and propagate. 8

3.2 X-axis shows the pyramid level and Y-axis the runtime tile search and propagate. 9

4.1 X-axis shows the pyramid level and Y-axis the runtime tile search and propagate. 12

4.2 X-axis shows the pyramid level and Y-axis the runtime tile search and propagate. 13

4.3 X-axis shows the pyramid level and Y-axis the runtime tile search and propagate. 14

4.4 Xaxis shows the pyramid level and Y-axis the runtime tile search and propagate. 15

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. . . 16

4.6 Power Efficiency of Camera ISP Stages in different configurations. 17

4.7 Framesize of I and P frames 18

5.1 X-axis shows the pyramid level and Y-axis the runtime tile search and propagate. 26

A.1 Six fisheye images as captured by the IMX274 using Jetson TX2 board. 35

A.2 Equirectangular Projection of first and second camera frames 36

A.3 Optical flow inputs: Overlapping regions of adjacent camera images. Equirectangular Projection of first and second camera frames 37

A.4 Cubemap based Stitching, Path : /media/gunman/Data/fall-2017/research/mono360/equi2cu

Chapter 1

INTRODUCTION

From the experiences of having see monoscopic 360 videos on samsung gear VR, I felt that complete potential of VR devices is not being utilized. Although the current âĂIJHead Mounted DevicesâĂİ devices are capable of showing 360 videos with depth information, we donâĂŹt have lot of content that is stereo 360. In order to make this happen we need simpler and smaller devices which should be portable for consumer use.

Current stereo 360 video systems are used mainly designed for professional videography. They consist of several camera(18 in googleâĂŹs jump VR) with bulky camera rig to support them. They record lot of data which will be offloaded and used to generate stereoscopic 360 videos.

This week, I started looking into literature on 360 degree systems. I found 3 works particularly interesting. In the first work âĂIJ6-DOF VR Videos with a Single 360-CameraâĂİ, they use âĂIJstructure from MotionâĂİ and video stabilization techniques to generate stereo 360 videos with 6DoF from a monoscopic 360 capture device. Other two works are from google and facebook respectively. GoogleâĂŹs work uses 16 cameras and facebook is trying to use just 4 cameras. Both of them provide different set of challenges. I am currently reading these two works and trying to figure out the direction for my thesis work. ———————————————————— The stitching pipeline optimization for quality have been published since last two decades. But it is mostly on software and they are very general algorithms and doesnâĂŹt consider power budget for implementing 360 capture using a mobile device. The image capture pipeline is very simple and is not much scope for novelty

as the single image frame stitching pipeline in hardware is straightforward. We need to think of smart ways to scale it for video stitching in 360, especially for making the videos spatio-temporal consistency between frames when the object lying on the boundary are moving across the fisheye images. ———————— Usecases See the commit change 2 Overview of paper

use cases - 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.

Status quo: Existing 360 camera solutions.

Characterizing monoscopic, stereoscopic, bottlenecks, optimization

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. 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

Chapter 2

BACKGROUND

Existing systems

Google Jump, Facebook Surround, Mega Stereo, Samsung Gear 360

Differentiating our work

Bottlenecks in the existing systems

Terminology: Pinhole Camera Fisheye Spherical Projection Equirectangular Projection Camera Calibration Process Intrinsics Extrinsics Optical Flow Temporal Regularization Pyramid Video Compression: Stages in 3D video recording: Capture Data compression and transfer Video frame Queue Stitching precomputing GPU vertex shader GPU fragment shader Processed queue Encoding

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

YâĂŹCbCr is used to mainly for storage and transmission efficiency. YâĂŹ - 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 YâĂŹ is used with high precision and Cb Cr can be of low precision. ———————————— Compression Related Work: Conduit en-

coding: 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

MOnoscopic Stages: The stages may include Intensity compensation, fisheye unwarping, alignment, blending. The main goal of this work is to reduce bandwidth requirement and remove redundant computation during different stages.

Data Flow

Block Diagram - With different stages. - With Data Inputs and Data Outputs of each stage. (Zoom in Diagram)

Motion detection

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.) : 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 GPUâĂŹs)

Facebook also has 360 stereo solutions by name âĂIJfacebook surroundâĂİ. One of them with 24 cameras and other with 6 cameras. I may need to find data for them

as well.

The power numbers says lot of scope for optimization using the techniques we are going to propose. My work for next one week will be to understand the math for generating the two views for the stereo.

Chapter 3

SYSTEM OVERVIEW AND DATA FLOW

[Intro Paragraph here.]

3.1 Prototype System Overview

The end to end camera system, as shown in figure 3.1 can be divided into 4 major stages by energy consumption, viz., image sensor, image signal processor(ISP), processor, and Off-chip memory. For our prototype camera design we use six IMX-274 cameras for capture and Nvidia Jetson TX2 for supporting camera capture and processing. The Camera and Jetson specs are shown in figure 3.2.

Prototype System Overview

Hardware: System 1) Six Camera Rig for OmniDrirectional Stereo

System 2) Dual Fisheye Camera for monoscopic 360

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

Data Flow

3.2 Data Flow

Block Diagram - With different stages. - With Data Inputs and Data Outputs of each stage. (Zoom in Diagram)

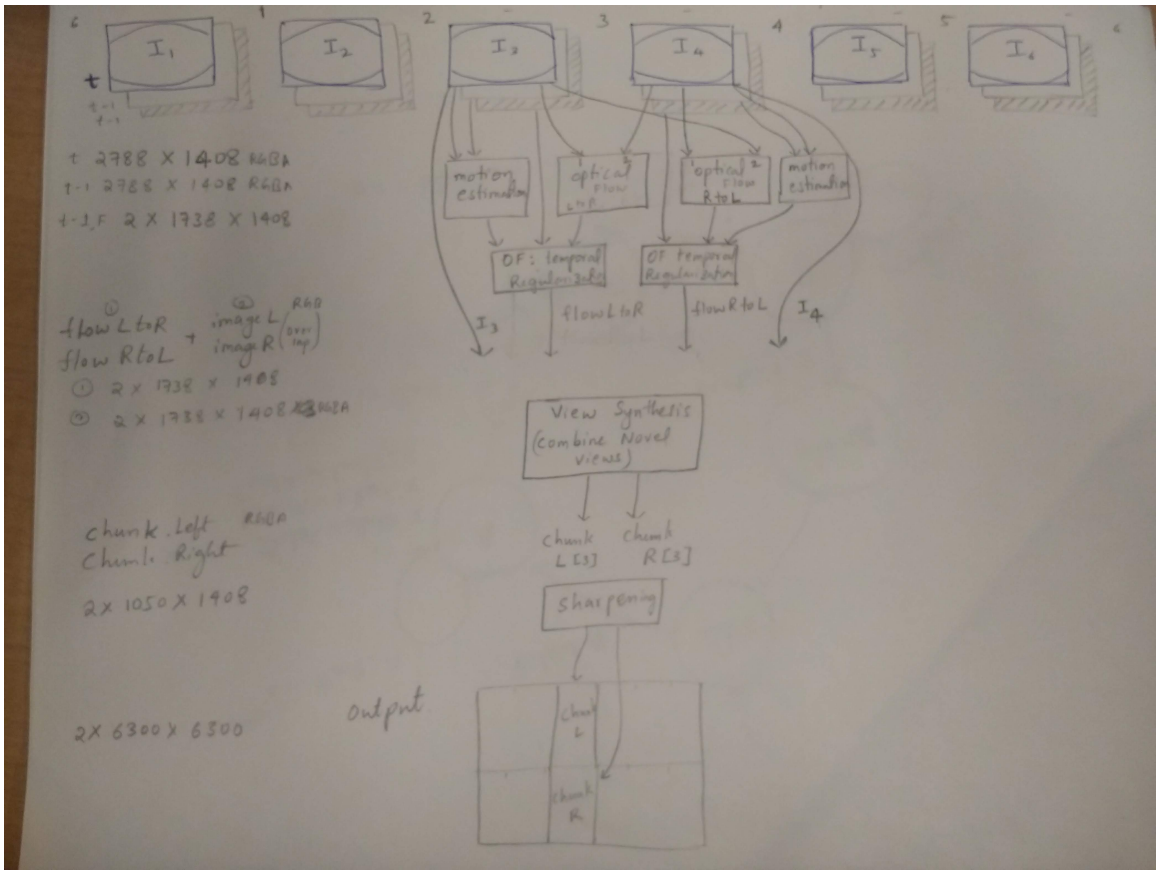

Figure 3.1: X-axis shows the pyramid level and Y-axis the runtime tile search and propagate.

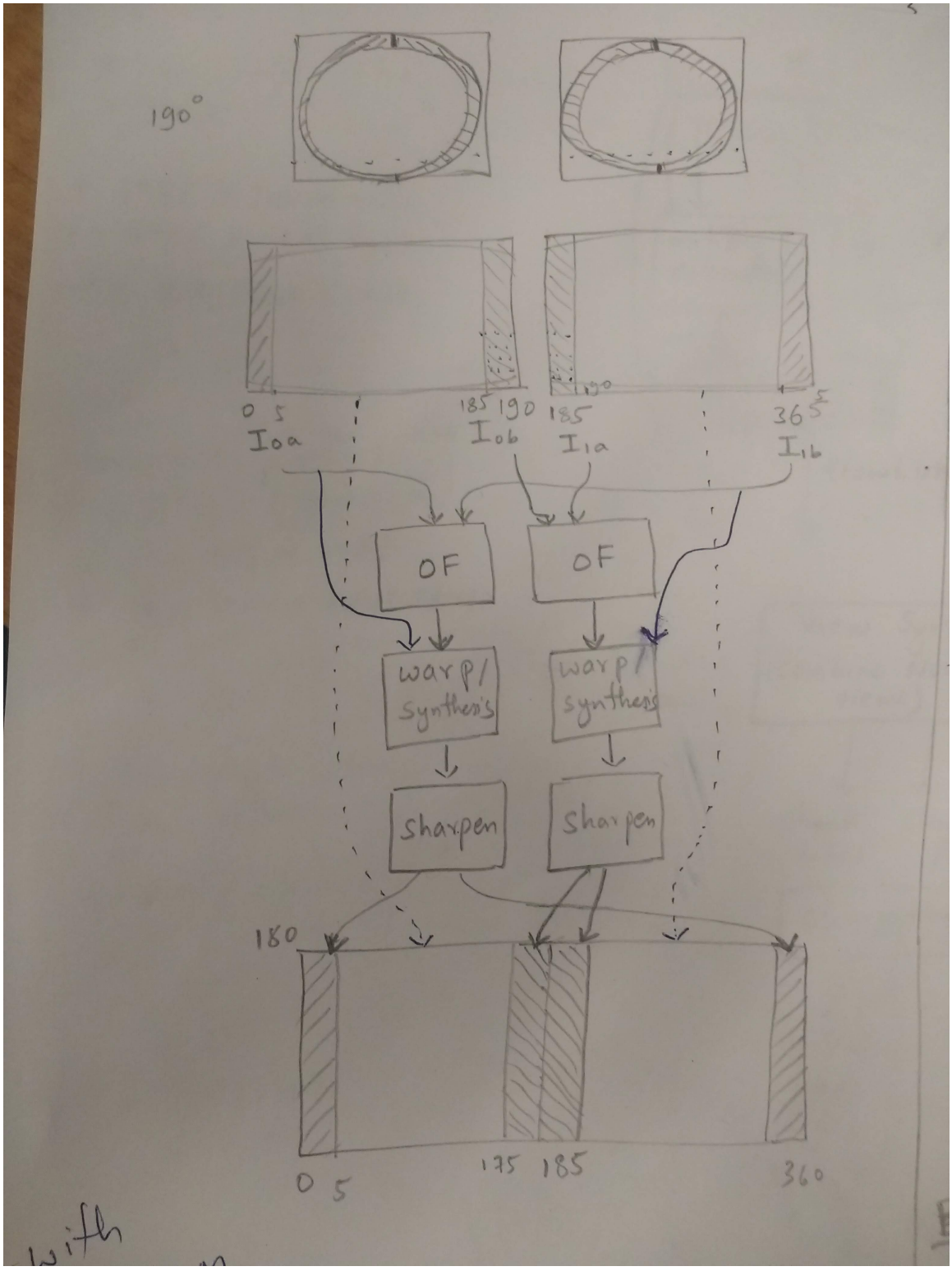

Figure 3.2: X-axis shows the pyramid level and Y-axis the runtime tile search and propagate.

Chapter 4

CHARACTERIZATION

Our work focuses on characterizing the energy and latency of end-to-end Omni-directional(OD) Camera systems. The goal is to find the bottlenecks of different components 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 we see a potential research opportunity to close gaps between hardware and software. 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. 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.

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.

We characterize both monoscopic and ODS camera systems. The difference between them is the number of novel views needed is significantly higher for ODS. Monoscopic is a special case of ODS and at core involves the same 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 power monitor IC and ways to moniter CPU, GPU, memory frequencies. We measure the absolute energy of the system and the difference between the idle and active state for individual stages of the pipeline. Measurement by experiments: Jetson power monitor for camera and ISP, and Optical flow power from Zynq Analytical and Simulation based: Micron System Power Calculator for LPDDR2. Application run time measurement.

4.2 Energy Characterization

4.2.1 Individual stage energy

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

The camera system is configured to 1920x1080 resolution at 30 fps for the below measurements. [make absolute energies instead of diff. i.e active -idle]

As seen in the above table, the processing the optical flow implementation in software is expensive both in terms of energy and latency. 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.

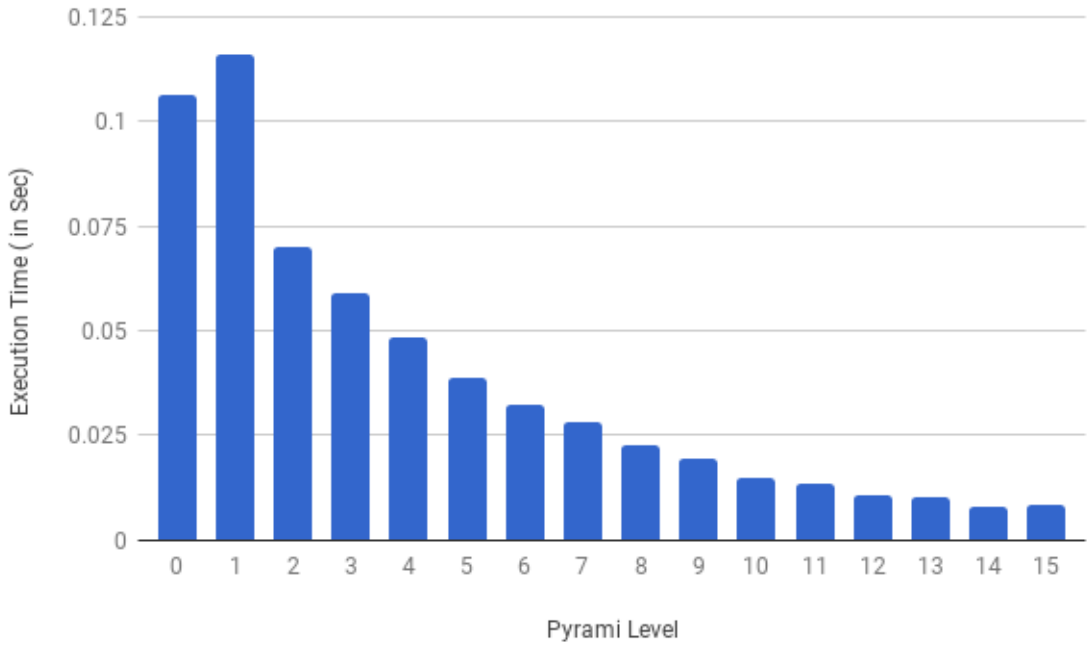

Figure 4.1: X-axis shows the pyramid level and Y-axis the runtime tile search and propagate.

4.3 Latency Characterization

Performance

a) Individual Stage Latency Characterization

4.3.1 Individual Stage latency

For camera system and ISP stages the latency is taken from NVIDIA camera API documentation. We measure the latency of computation in terms of CPU runtime.

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 the runtime tile search and propagate.

4.4 Sensor Response Time

Camera startup time needs to be less. Initial frames are low contrast but improves over time.

4.5 Design Scalability

Runtime scalability with the resolution

Outputs: 4k, 6k, 8k, 12k Discuss scalability of resolution for different stages. Especially the scalability of cache, DRAM, CPU power.

What is the energy per each output pixel What is energy per pixel when generating

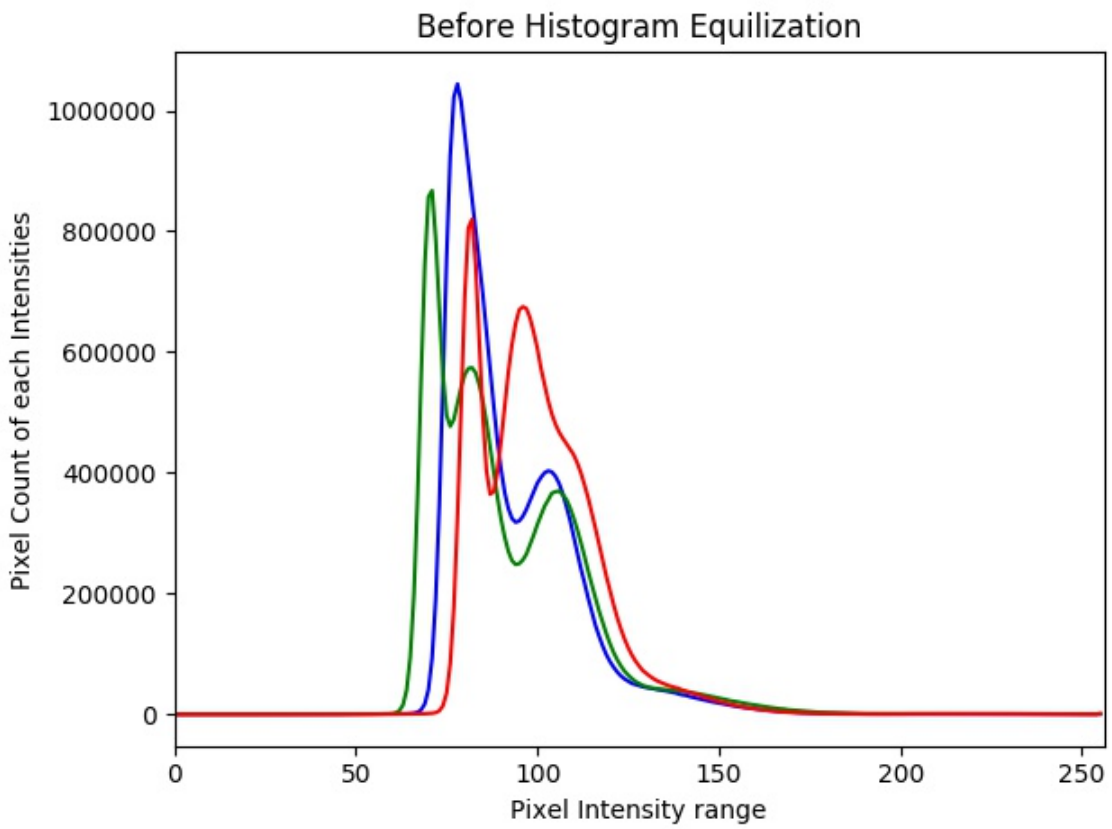

Figure 4.3: X-axis shows the pyramid level and Y-axis the runtime tile search and propagate.

only one ODS view, and how does it compare when generating two views! If it is double then we have a problem to solve [Check why sharpening is so costly!]

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. For 3k, 4k, 6k and 8k output resolution.

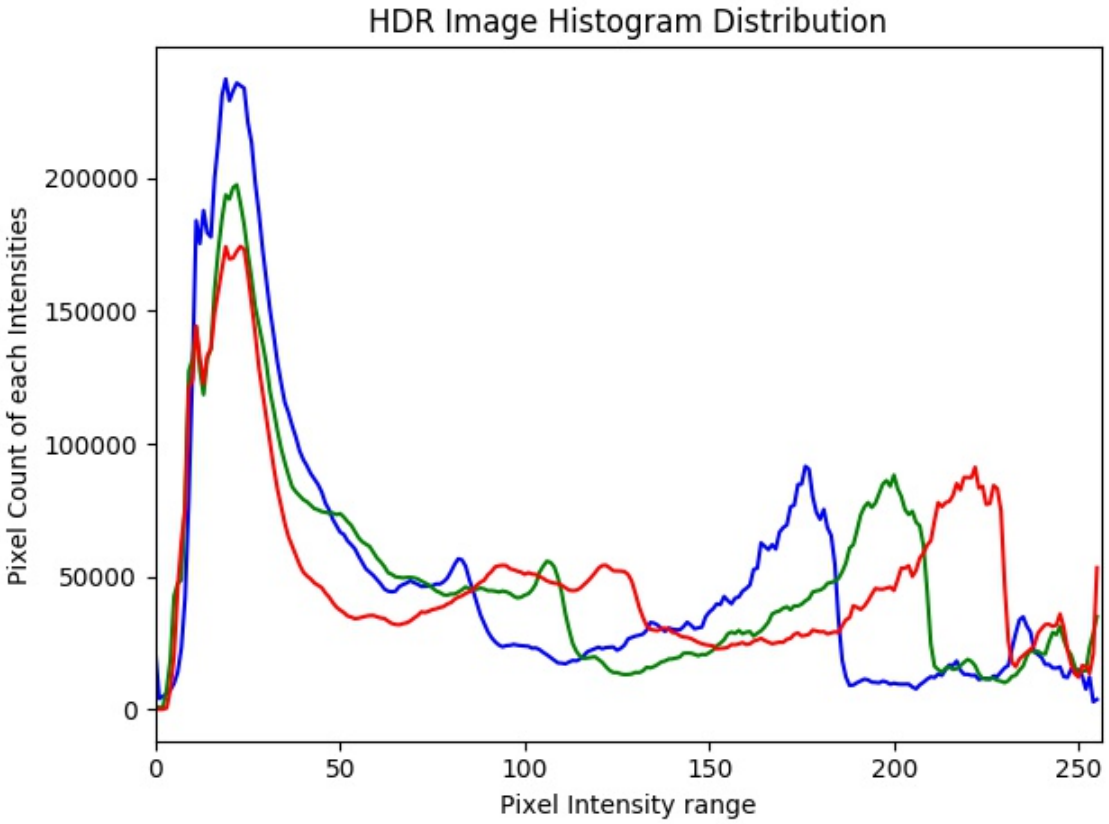

Figure 4.4: Xaxis shows the pyramid level and Y-axis the runtime tile search and propagate.

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

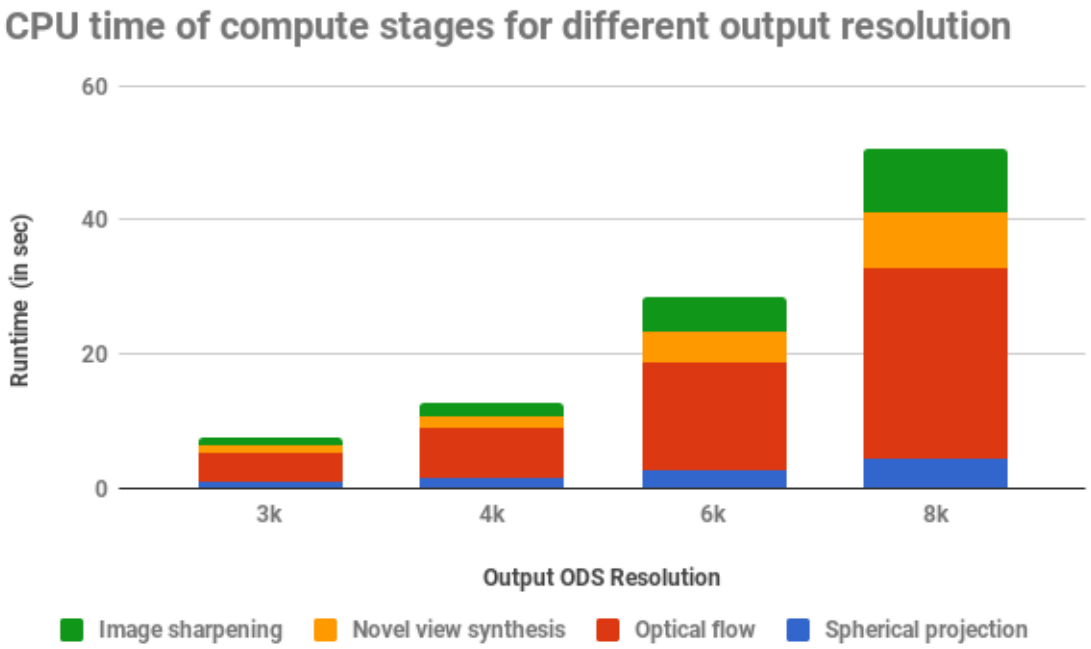

Figure 4.5: CPU execution time of different compute stages. X axis has different sub-stages in optical flow and Y axis correspond to energy per frame.

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	-	-	-	-

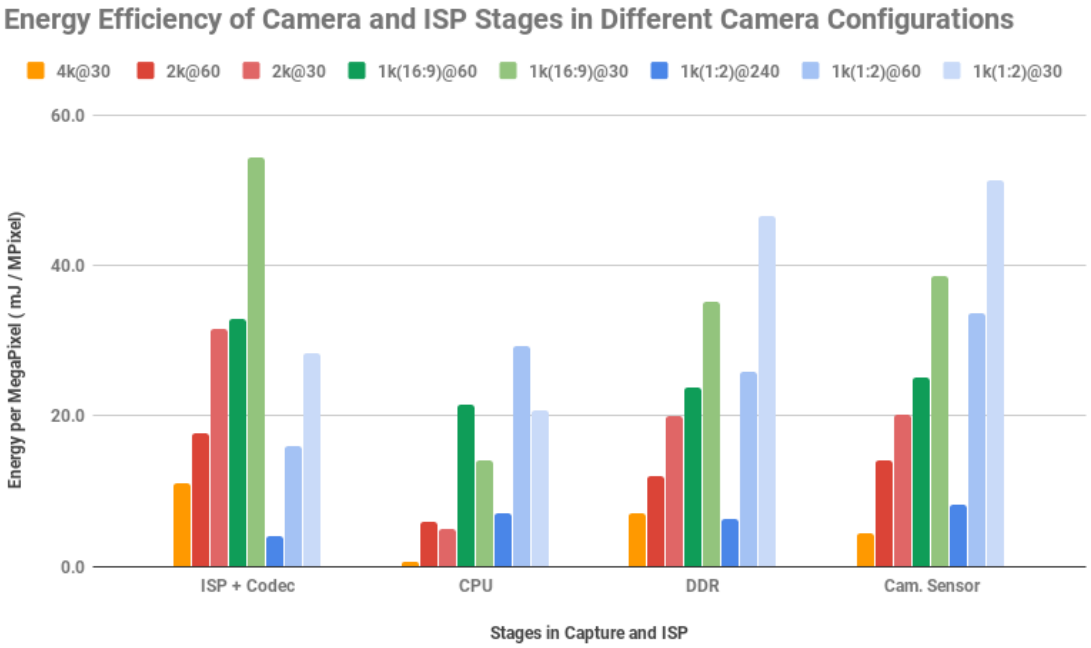

Figure 4.6: Power Efficiency of Camera ISP Stages in different configurations

Static and Dynamic Power. Resourse Utilization. Clock gating: We were able to do clock gating of pixel clock. It works fine. The next step is to select between multiple clocks and see how it affect the output image.

Latency: By varying the latency of reconfiguration, the image is not stable. It shows horizontal and vertical bars of image. This has to be fixed in this week.

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 with computation time to save energy and latency. It also shows that the accuracy

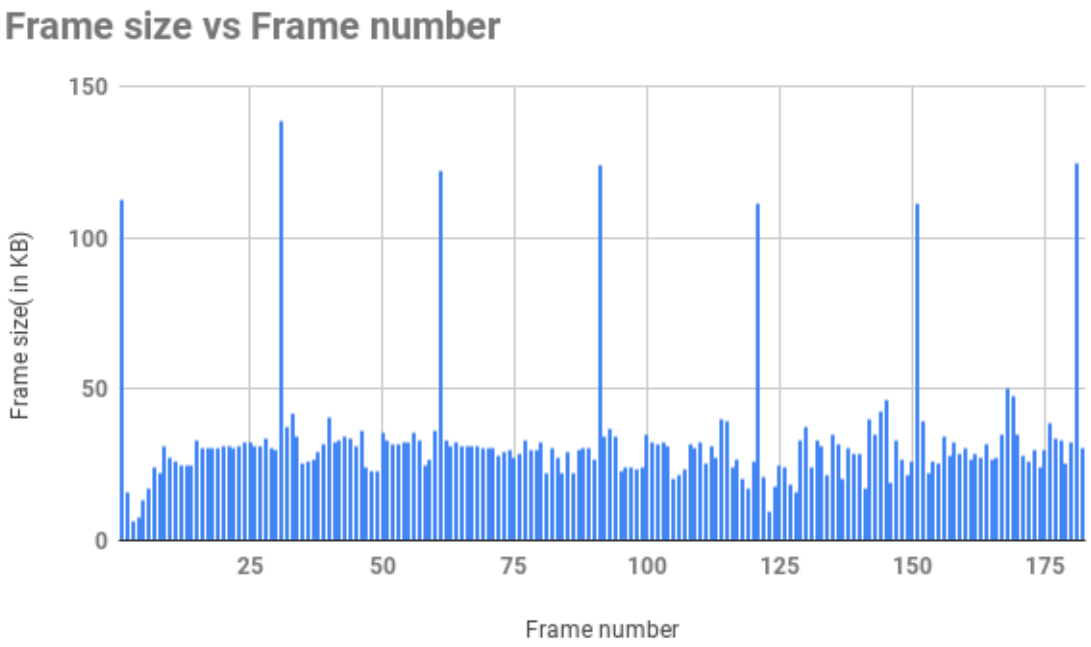

Figure 4.7: Framesize of I and P frames

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.

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 360Âžx180Âž 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 360Âž 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

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

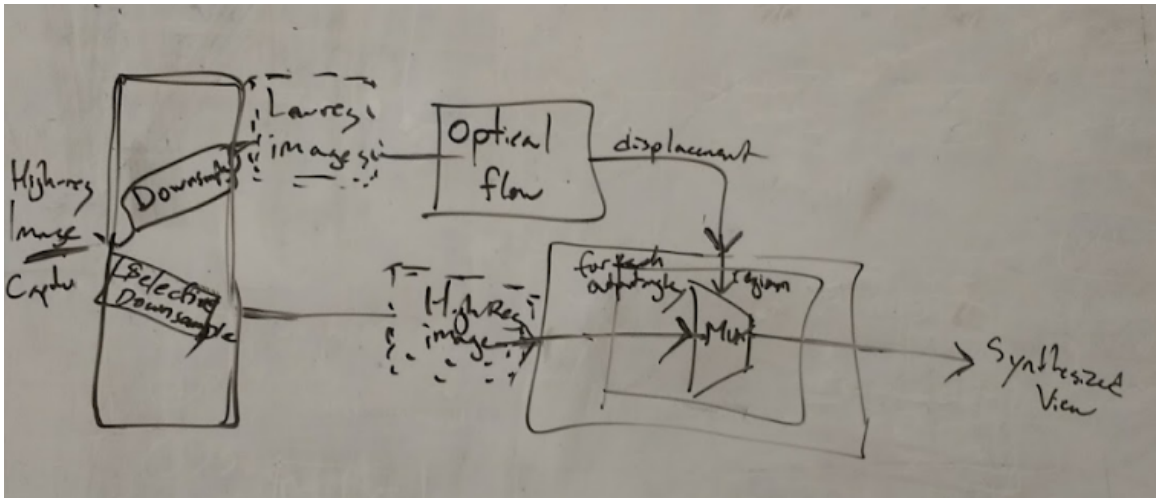

Figure 5.1: X-axis shows the pyramid level and Y-axis the runtime tile search and propagate.

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.

Chapter 6

DISCUSSION AND CONCLUSION

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 donâĂŹt 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 Section2-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 wonâĂŹt 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

Chapter 7

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?

7.1 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] doesnâĂŹt 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.

Chapter 8

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.
The overlapping left and right images.

Maximum Function For Extreme low power applications: Since the alignment techniques is not the final algorithm I may be going ahead with, I want to start with something even simpler, i.e a maximum function. We can imagine the core computation block as a black box, that can be changes as per the needs of the application. This becomes an interesting use case as the final camera could have multiple such black boxes which can be turned on and off based on application demand. For eg. If the camera is being used for CNNs, maximum function might be good enough. If itâĂŹs going to be used for scene capture, then we can turn on an efficient stitching core instead of maximum function.

Figure A.1: Six fisheye images as captured by the IMX274 using Jetson TX2 board.

v3.jpg

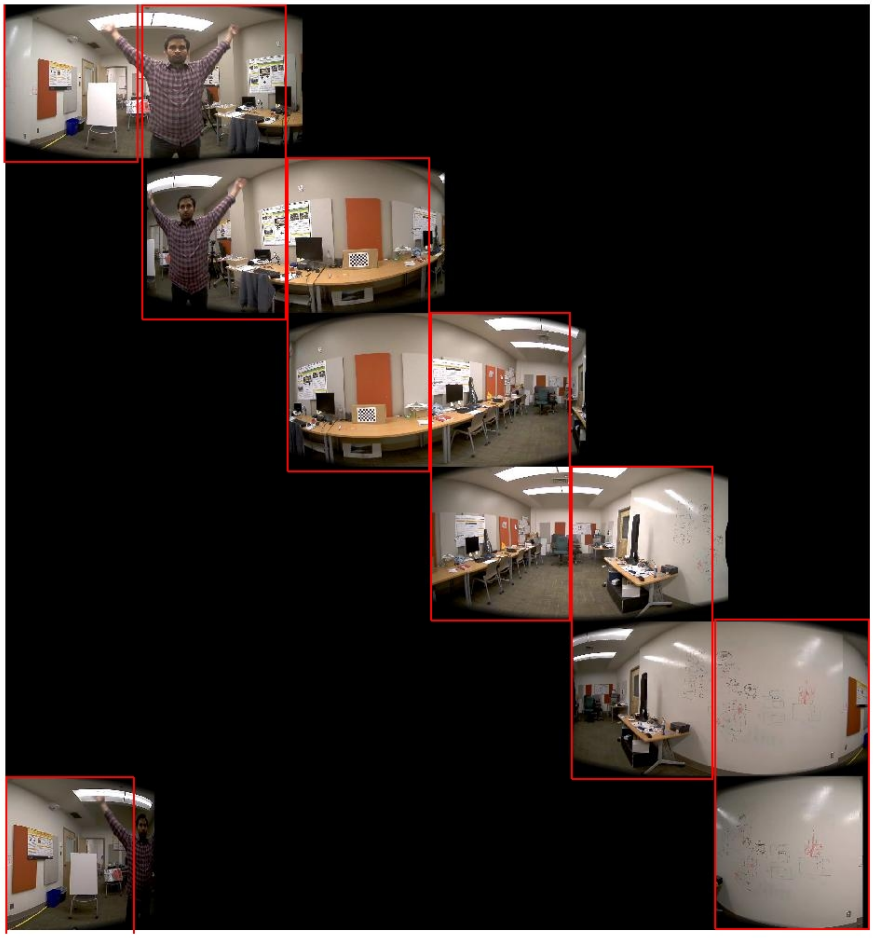

Figure A.2: Equirectangular Projection of first and second camera frames

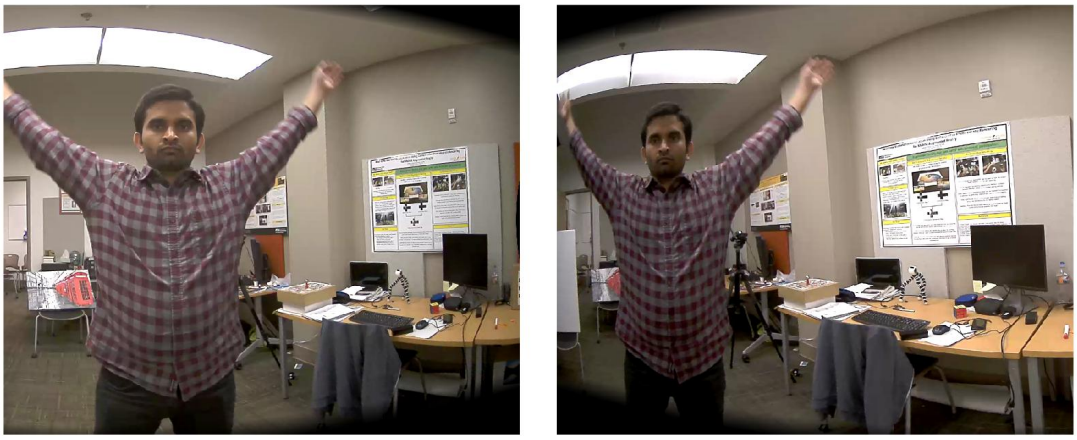

Figure A.3: Optical flow inputs: Overlapping regions of adjacent camera images. Equirectangular Projection of first and second camera frames

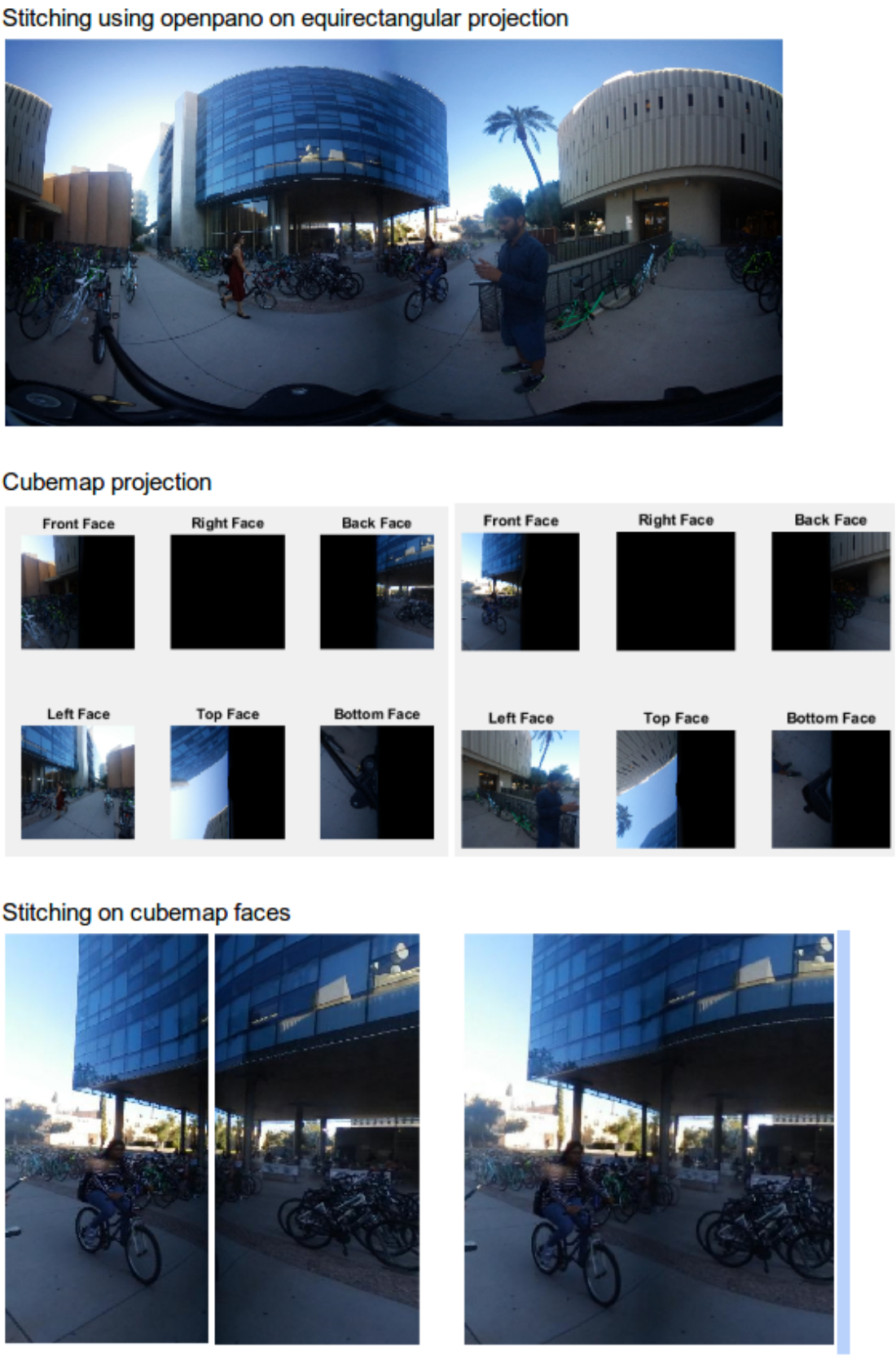

Figure A.4: Cubemap based Stitching, Path : /media/gunman/Data/fall-2017/research/mono360/equi2cubic

APPENDIX A

MAKING OF THE CAMERA RIG

Camera Calibration Process OCamCalib

APPENDIX A

SYSTEMS ARCHITECTURE SURVEY

Heteregeneous Systems Memory Access scheduling has been very critical since the single core processor. [5] Rixner et. al. introduced the scheduling of DRAM operations out of order, to optimize memory system performance. They reordered the memory references to exploit row buffer locality. But the major drawback of this kind of memory schedulers was they optimized for throughput rather than for the overall performance of different application threads in the system. This became more evident with the chip multiprocessors(CMP). Mutlu and Moscibroda proposed a Stall Time Fair Memory(STFM) Access Scheduling[4] to overcome these issues. STFM provides quality of service(QoS) to all the threads by measuring the slowdown of threads. The results show that slowdown related to thread level interference is significantly reduced and also, it improved average system throughput.

In the last few years, hardware accelerators have dominated the SoC from mobile to High-performance computing. The modern mobile SoCâĂŹs[7, 8] are heterogeneous architectures that integrate CPU with hardware accelerators like GPU, image processor, audio processor. On the other hand, High-performance Computing clusters are now being accessible through cloud services like amazon web services[9] and others. Heterogeneous System Architecture(HSA) [11] is being developed by multiple companies to integrate CPU and GPU unified virtual memory. As a step further Unified memory is currently introduced in Nvidia CUDA 6. All of this motivates to further study on application interference, meeting QoS targets for close integration of CPUs and GPUs.

The heterogeneous systems are currently being optimized to have same off-chip memory and even share ldast level Cache to reduce area, remove memory copy overheads and for tighter integration between CPU and Hardware Accelerators(HWA). The sharing of resources lead to interference in memory scheduling if not prioritized efficiently. Since the heterogeneous systems are required to provide support to multiple application, QoS of each application becomes very critical. Several works have proposed in this area[1,2,3,12]. Staged memory scheduling [1] proposes a new approach to improve system performance and fairness, especially in integrated CPU-GPU systems. Although it provides fairness, it doesnâĂŹt take care of strict QoS requirements by applications. Jeong et al.[12] prioritize applications when the HWAâĂŹs are unable to meet the application deadlines. This mechanism doesnâĂŹt help resolve the QoR issues completely as the applications don't meet the deadline when they are prioritized very late. Dash, Squash [2,3] propose ideas to overcome this. Their first idea, Distributed Priority, prioritize the HWA when they are distant from the deadline, thereby removing the risk of missing the deadline. Their second idea is application aware scheduling for CPUâĂŹs. This is based on the observation that memory intensive CPU applications are not affected when HWAâĂŹs are given priority over them. The third idea is to prioritize HWAâĂŹs with a short deadline with the highest priority when they are close to the deadline.

The DASH and SQUASH schemes mainly focus on applications where there are multiple kinds of HWAâĂŹs, and there is contention for memory resources. This kind

of systems is common in mobile systems. But there is another breed of heterogeneous systems which is dominated by GPUâĂŹs and CPUâĂŹs, mainly used in high-performance computing(HPC). The ideas given in Dash and SQUASH cannot be directly applied to HPC systems. Through this project, I plan to study the heterogeneous systems dominated by GPUâĂŹs. The characterization studies by [13, 14] make observations related to CPU-GPU systems. [14] evaluates the performance seen with coarse grain and fine grain synchronization. They test two types of memory configurations, i.e separate L3 cache vs shared L3 cache. [13] concludes that shared LLC will provide better caching for heterogeneous processors compared to discrete GPU systems. So my project will focus on evaluating the performance bottlenecks and the meeting the QoS requirements for CPU-GPU based heterogeneous systems where Last Level cache is shared.

References:

Experience With FPGA Vs NVIDIA Jetson:

Last weekâĂŹs target is to complete the Alexnet CNN layer C1 to C5 operations using DRAM based FPGA implementation. Since the run time of the tools is in the order of hours and even days(for place and route step), I planned my work as follows. I have divided my work into two parts. First one is to take simple accelerator design(AXI, AXI-lite based simple accelerator + Microblaze) and perform IP integration, generating the drivers files. Xilinx SDK will then be used to write c program using driver APIâĂŹs to transfer data between microBlaze and AXI-based memory-mapped I/O interface of the simple accelerator. The second task is to take the actual design C1-C5 layers of Alexnet through same steps. The first task is successful. I used Xilinx SDK tools to write simple c-codes to transfer data between microblaze and accelerator. I used axi-lite interface of the accelerator to send and receive the data using microblaze.

The second task is to take the actual design through the same design flow and use drivers APIâĂŹs to do data transfer. Before doing that I needed to rewrite the c++ code to make sure that all the data required by CNN accelerator can be received by single AXI interface(burst mode). I changed it so that accelerator will get the start pointer of array, and control signals(start, stop) through axi-lite interface and then the accelerator will make memory requests using AXI burst mode at high data rates. I did these changes and I am going through place and route step currently.

The tool run-time has been a big problem currently. It takes forever to complete synthesis/implementation steps. The baseline design(Microblaze + simple addition accelerator) takes 1 hour time. But the runs with actual Alexnet accelerator + microblaze doesnâĂŹt complete even after day. The runs for just C1 layers hasnâĂŹt run to complete even after one full day. I had to plan well, so that IâĂŹm not idle when the runs are happening. For example, I would work on the design interfaces, integration of IPâĂŹs, validation, and development of application in SDK, when the runs are happening.

Although there are runtime issues, others things are either figured out or in the process of implementation. I got to know about the entire design flow and integrating accelerators, interfacing different IP modules using AXI, and hardware/software development for embedded systems. This should make my work easier for future

projects that involve fpga development.