DESIGN AND EVALUATION OF EXTENDED REALITY SYSTEMS

BY

MUHAMMAD HUZAIFA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

Doctoral Committee:

Professor Sarita Adve, Chair
Associate Professor Girish Chowdhry
Assistant Professor Christopher Fletcher
Professor Kris Hauser
Professor Steve LaValle
Dr. Armin Alaghi, Meta Reality Labs Research
Professor David Brooks, Harvard University

ABSTRACT

As we enter the era of domain-specific architectures, systems researchers must understand the requirements of emerging application domains. Augmented and virtual reality (AR/VR) or extended reality (XR) is one such important domain that can propel new directions in architecture and systems research, as well as have a profound impact on all aspects of human endeavours. Thus far, conducting end-to-end systems research in XR, and consequently, developing principled design methodologies, has remained a challenge due to a lack of open-source benchmarks and systems. To address this challenge, this thesis develops a novel open-source XR research testbed, proposes several design principles for optimizing the power consumption of XR systems, and demonstrates the efficacy of these design principles in optimizing key XR components.

First, we present ILLIXR, Illinois Extended Reality Testbed, the first fully open-source end-to-end XR system with state-of-the-art components, integrated with a modular and extensible multithreaded runtime, providing an OpenXR compliant interface to XR applications (e.g., game engines), and with the ability to report (and trade off) several quality of experience (QoE) metrics.

Next, we analyze performance, power, and QoE metrics for the complete ILLIXR system and for its individual components, the first such analysis to be publicly available. Our analysis reveals several properties with implications for architecture and systems research. These include demanding performance, power, and QoE requirements, a large diversity of critical tasks, inter-dependent execution pipelines with challenges in scheduling and resource management, and a large tradeoff space between performance/power and human perception related QoE metrics.

We then explore several broadly applicable design principles for optimizing the power consumption of XR systems. We demonstrate the use of distributed on-sensor computing for optimizing head tracking, which reduces VIO subsystem power by 1.3× compared to a centralized baseline. We employ another principle, online frequency control, to optimize scene reconstruction by reducing its frequency, resulting in a 4.9× reduction in power compared to a state-of-the-art baseline. We further reduce the power consumption of scene reconstruction by using scalable hardware mapping, which results in an overall power reduction of 128×–270× compared to an embedded GPU. The end-to-end nature of ILLIXR has enabled research in compute offload, new networking technologies, QoE-driven cross-component approximate computing, QoE-driven scheduling, and automatic accelerator detection.

Overall, this thesis presents the research community with a unique testbed that enables architecture and systems research in XR in particular, and domain-specific systems in general. It presents a characterization of the testbed, providing researchers a starting point for conducting XR research. It also presents several generally applicable design principles for optimizing power, and demonstrates their use in optimizing head tracking and scene reconstruction.

بِسْمِ ٱللَّهِ ٱلرَّحْمَـٰنِ ٱلرَّحِيمِ

In the name of Allah, the Most Gracious, the Most Merciful

To Mama and Baba, for their enormous sacrifices
To my grandfather, for changing the fate of generations to come

ACKNOWLEDGMENTS

I would like to start by thanking my Lord and Creator, Allah. He has given me everything that I have, and no words or praise can describe my gratitude to Him for bestowing upon me His innumerable Blessings and Bounty.

It took a village to raise this particular PhD student. I will not be able to do justice to each and every single person in this village in just a few pages, but I will do my best.

The head of this village was my advisor, Professor Sarita Adve. I never quite understood why people thanked their advisors before anyone else in their theses. I used to think that no matter how important an advisor was, they were ultimately not at the same level as ones parents and other loved ones. It turns out that I was wrong. I am forever indebted to Sarita, for she has had a profound impact on my life, both academically and otherwise. Sarita took me on as an advisee when I was a wide-eyed, naive student, and nurtured me into the researcher that I am today. She helped cultivate all the skills that I have, and inculcated a passion and work ethic in me that I am sure will serve me my entire life. I thoroughly tested her patience over the years, and I am grateful for how she responded with support and kindness. It was a joy to go through the ups and downs of ILLIXR with her. Thank you, Sarita, for everything.

Next, I would like to thank my committee members, my interactions with whom have taught me many lessons. Professor Girish Chowdhry, for showing me how engineering is done quite literally in the wild. Professor Christopher Fletcher, for bringing an energizing enthusiasm to our few, but important meetings. Professor Kris Hauser, for forcing me to think about the big picture. Professor David Brooks, for validating that remaining cool, calm, and composed is *always* a good thing. Professor Steve LaValle, for helping us jump start ILLIXR with his breadth and depth of knowledge, and for inspiring a passion in me to not only make extended reality a reality, but also to share what I learn with those in my community in a fun and exciting way. Dr. Armin Alaghi, for giving me an internship at Meta in 2019, mentoring me, and cold emailing people to introduce me to them. Many of those people played important roles in the development of ILLIXR, and I am thankful to Armin for investing his time in me.

My work was supported in part by the National Science Foundation under grants CCF 16-19245, CCF 21-2046, and 2217144, by the Center for Future Architectures Research (C-FAR), one of the six centers of STARnet, a Semiconductor Research Corporation (SRC) program sponsored by Microelectronics Advanced Research Corporation (MARCO) and Defense

Advanced Research Projects Agency (DARPA), by the Applications Driving Architectures (ADA) Research Center, a Joint University Microelectronics Program (JUMP) Center co-sponsored by SRC and DARPA, by the DARPA Domain-Specific System on Chip (DSSoC) program, by a Sohaib and Sara Abbasi Fellowship, and by a Google Faculty Research Award. I performed a portion of this work while I interned at Meta.

I would like to thank Dr. Wei Cui, Dr. Aleksandra Faust, Professor Liang Gao, Rod Hooker, Dr. Steve Lovegrove, Dr. David Luebke, Dr. Andrew Maimone, Vegard Øye, Maurizio Paganini, Dr. Martin Persson, Dr. Archontis Politis, Professor Eric Shaffer, Professor Paris Smaragdis, and Professor Shenlong Wang, for their advice and help. I would like to especially thank Dr. Ameen Akel, Dr. Matt Horsnell, and Amit Jindal, for supporting and believing in ILLIXR from day one.

Next, I would like to thank Dr. Chris Widdowson, for patiently helping me build foundational knowledge in XR; Dr. Santosh Abraham and Dr. Wilson Fung, for teaching me much of what I know about graphics, simulators, and software engineering; Professor Warren Gross and Professor Brett Meyer, for being amazing undergraduate advisors; Viveka Kudaligama, Kathy Ann Runck, Glen Rundblom, and the rest of the academic office and Engr-IT, for being the unsung heroes of the department; and the various doctors in Urbana-Champaign and Pakistan, for nursing me back to health over the years.

I am very grateful for having amazing lab mates and collaborators, all of whom I would like to thank: Dr. John Alsop, Jumbo Jiang, Qinjun Jiang, Ying Jing, Dr. Rakesh Komuravelli, Jae Lee, Fang Lu, Bakshree Mishra, Yihan Pang, Joseph Ravichandran, Giordano Salvador, Bill Sherman, Finn Sinclair, Professor Matt Sinclair, Rahul Singh, Vignesh Suresh, Boyuan Tian, Professor Radha Venkatagiri, Hengzhi Yuan, and Jeffrey Zhang. I particularly enjoyed my conversations with The Council of Elrond: Wes Darvin, Rishi Desai, and Samuel Grayson.

Outside of work, I have been very fortunate to have great friends. I would like to thank Mohammad Ahmad, Wajih Ul Hassan, Saad Hussain, Shalan Naqvi, and Hassan Shahid, for making 61801 fun and memorable (and for almost derailing my PhD); Athar Ejaz, Usama Haq, Umer Shahzad, and Ali Qureshi, for sticking around all these years; Hashim Sharif and Hareem Dar, for all the great conversations; Rashid Tahir, for aiding me in my metamorphosis; and Owais Khan, for all the deep discussions and for giving sound advice when I needed it most.

A very special thank you goes to a bunch of very special people who have gone above and beyond in taking care of me: Zeeshan Fazal and Sana Zeeshan, for all the laughs and for making me feel like a part of their family; Gohar Irfan and Qandeel Tariq, for "always stepping forward" and for flying over to CU in my time of need; and Samir Khan, Salihah

Aleem, and Musa Khan, for giving me a place to stay and making my PhD, especially the last few months, enjoyable and comfortable.

A most special thank you goes to Dr. Abdulrahman Mahmoud, my brother from another mother, and someone who has been by my side since the day I landed in Urbana-Champaign. My brother, you have made everything from doing a PhD to riding motorcycles to playing video games to praying so much more fulfilling, and I deeply cherish our bond.

I have thanked several teachers and friends so far. But my oldest teachers and friends have been my family. My mother, Afia, and my father, Farrukh. I am indebted to them for always supporting me in all my endeavours, be they pursuing this PhD or jumping from a plane. My parents have made enormous sacrifices in their personal lives for their two children, and this dissertation is dedicated to them. Although it pains me to do so, I would also like to thank my brother, Yahya, for his witty banter and for always making me laugh.

Finally, I also dedicate this dissertation to my grandfather, Dr. Chiragh Muhammad Khan. He was a teenager when the partition of India happened in 1947. He worked on a farm to support his family, and made the effort to become an orthopedic surgeon to raise himself and those around him from grinding poverty. I would never have had the chance to do this PhD if it were not for him and everything he has done since 1947. Thank you, dada abbu.

TABLE OF CONTENTS

دیارِ عشق میں اپنا مقام پیدا کر
نیا زمانہ، نئے صبح و شام پیدا کر

خدا اگر دلِ فطرت شناس دے تجھ کو
سکوتِ لالہ و گل سے کلام پیدا کر

مِرا طریق امیری نہیں، فقیری ہے
خودی نہ بیچ، غریبی میں نام پیدا کر

CHAPTER 1: INTRODUCTION

Recent years have seen the convergence of multiple disruptive trends to fundamentally change computer systems: (1) With the end of Dennard scaling and Moore's law, application-driven specialization has emerged as a key architectural technique to meet the requirements of emerging applications, (2) computing and data availability have reached an inflection point that is enabling a number of new application domains, and (3) these applications are increasingly deployed on resource-constrained edge devices, where they interface directly with the end-user and the physical world.

To truly achieve the promise of efficient edge computing, however, will require architects and system designers to broaden their portfolio from specialization for individual accelerators to understanding domain-specific *systems*. Such systems may consist of multiple interacting sub-domains, requiring multiple accelerators that interact with each other in myriad ways to collectively meet end-user demands and quality of experience (QoE) metrics. Programming languages and OS researchers must also grapple with the heterogeneity of such domain-specific systems, developing scalable methods for compilation, scheduling, and resource management. Furthermore, it is likely that meeting the end-user quality demands of such systems will require co-designing the hardware, compiler, and OS along with the application [1, 2].

The emerging domain of virtual, augmented, and mixed reality, collectively referred to as extended reality (XR),[1] is a rich domain that can propel research needed for this era of end-to-end quality-driven, resource-constrained, co-designed domain-specific edge systems. This claim rests on the following observations:
(1) Pervasive: XR will pervade most aspects of our lives — it will affect the way we teach, conduct science, practice medicine, entertain ourselves, train professionals, interact socially, and more. Indeed, XR is envisioned to be the next interface for most of computing [3, 4, 5, 6].
(2) Challenging demands: While current XR systems exist today, they are far from providing a tetherless experience approaching perceptual abilities of humans. There is a gap of several orders of magnitude between what is needed and achievable in performance, power, and usability, giving architects and system designers a potentially rich space to innovate.
(3) Multiple and diverse components: XR involves a number of diverse sub-domains — video, graphics, computer vision, machine learning, optics, audio, and components of robotics — making it challenging to design a system that executes each one well while respecting the

[1] Virtual reality (VR) immerses the user in a completely digital environment. Augmented Reality (AR) enhances the user's real world with overlaid digital content. Mixed reality (MR) goes beyond AR in enabling the user to interact with virtual objects in their real world.

resource constraints.

(4) Full-stack implications: The combination of real-time constraints, complex interacting pipelines, and ever-changing algorithms creates a need for full stack optimizations involving the hardware, compiler, operating system, and algorithm [2].

(5) Flexible accuracy for end-to-end user experience: The end user being a human with limited perception enables a rich space of accuracy-aware resource trade-offs, but requires the ability to quantify impact on end-to-end experience.

A key obstacle to architecture research for XR is that there are no open-source benchmarks covering the entire XR workflow to drive such research. While there exist open source codes for some individual components of the XR workflow (typically developed by domain researchers), there is no integrated suite that enables researching an XR system. This is due to two reasons. First, developing an XR system requires expertise in a large number of sub-domains, such as graphics, computer vision, optics, audio, and video, among others, making it difficult to build a system.

Second, until recently, commercial XR devices had proprietary interfaces. For example, the interface between an Oculus head mounted device (HMD) system or runtime and the Unity or Unreal game engines that run on the HMD has been closed as are the interfaces between the different components within the HMD runtime. Consequently, an XR runtime developer would have had to devise their own custom application programming interface (API), and then implement calls to that API in each game engine or application that they had wished to use. While possible in theory, this approach was not scalable due to both the upfront and ongoing engineering effort required.

OpenXR [7], an open standard, was released in July 2019 to partly address this problem by providing a standard for XR device runtimes to interface with the applications that run on them. OpenXR has seen widespread adoption by both XR runtime developers, including Oculus, Microsoft, and Steam, and by game engine developers, such as Unity, Unreal, and Godot. As a result, it is now possible to design an XR runtime that is compatible with most applications by conforming to the OpenXR API . However, certain challenges remain as OpenXR is still evolving and does not yet address the problem of interfaces between the different components *within* an XR system.

In this thesis, we take the first step towards enabling researchers and system designers to perform end-to-end systems research in XR by developing ILLIXR—Illinois Extended Reality Testbed—the first open-source XR runtime. In contrast with previous and existing efforts, such as Monado [8], ILLIXR is end-to-end, OpenXR-compatible, and modular, with well-defined interfaces, making it well-suited for research, development, and benchmarking. Next, we use ILLIXR to perform the first publicly available analysis of a full XR system and

its constituent components, giving researchers insights and directions for future research. Finally, we show that optimizing XR systems will require the application of a multitude of *broadly applicable* design principles, with each principle potentially working for a family of components which share certain properties. The space of these design principles is vast, and in this thesis we focus on three such principles.

The first design principle that we demonstrate is distributed on-sensor computing, which we use to optimize the power consumption of head tracking. The second principle that we demonstrate is online frequency control, which we use to reduce the power consumption of scene reconstruction. The third and final principle we demonstrate is scalable hardware mapping, which we apply to scene reconstruction to further reduce its power consumption. Other possible principles include compute offload and cross-component approximate computing, both of which we explore in follow-on work, and summarize in Sections 1.7.1 and 1.7.2, respectively. The contributions of this thesis are summarized below.

1.1 A FULL-SYSTEM TESTBED FOR EXTENDED REALITY

To enable researchers to conduct end-to-end XR systems research, we develop ILLIXR (pronounced elixir)[2], the Illinois Extended Reality Testbed [9, 10] in Chapter 2. ILLIXR is the first open-source XR full-system testbed, consisting of (i) a representative XR workflow with state-of-the-art components, (ii) integrated with a modular and extensible multithreaded runtime, (iii) providing an OpenXR compliant interface to XR applications (e.g., game engines), and (iv) with the ability to report (and trade off) several QoE metrics. The ILLIXR components represent the sub-domains of robotics (Visual-Inertial Odometry or SLAM), computer vision (scene reconstruction), machine learning (eye tracking), image processing (camera), graphics (asynchronous reprojection), optics (lens distortion and chromatic aberration correction), audio (3D audio encoding and decoding), and displays (computational holography). We have open sourced ILLIXR,[3] and formed an industry-backed consortium to make ILLIXR a reference research testbed for the community.[4] We discuss in Section 1.7 how it is being used by others inside and outside Illinois.

[2]Samuel Grayson led the design and implementation of the runtime architecture.

[3]`https://illixr.github.io`

[4]`https://illixr.org`

1.2 ANALYSIS OF A FULL XR SYSTEM

We analyze performance, power, and QoE metrics for ILLIXR at the system level on desktop and embedded class machines with CPUs and GPUs, driven by a game engine running representative VR and AR applications [9, 10] in Chapter 3. This is the first such published analysis of its kind. We show that 1) current systems are far from the needs of future devices, 2) no single component dominates all metrics of interest (performance and QoE), 3) SoC and system logic, including data movement for displays and sensors is a major component of total power, and 4) most components exhibit significant variability in per-frame execution time due to input-dependence or resource contention. These findings make the case for specializing the system as a whole via software-hardware co-design.

1.3 CHARACTERIZATION OF XR COMPONENTS

In addition to performing a system-level analysis of ILLIXR, we also analyze each individual component of ILLIXR [9, 10] in Chapter 4. We show that XR components are quite diverse in their use of CPU, GPU compute, and GPU graphics, and exhibit a range of IPC and system bottlenecks. Analyzing their compute and memory characteristics, we find a variety of patterns and subtasks, with none dominating. The number and diversity of these patterns poses a research question for the granularity at which accelerators should be designed and whether and how they should be shared among different components. We summarize our approach for hardware reuse in Section 1.6.

These observations also motivate research in automated tools to identify acceleratable primitives, efficient on-chip memory hierarchies for connecting accelerators together, architectures for communication between accelerators, and accelerator software interfaces and programming models. We discuss a tool that aims to automate acceleration identification in Section 1.7.4 and Section 9.5. We discuss work on accelerator communication interfaces driven by ILLIXR in Section 9.7.

1.4 DISTRIBUTED ON-SENSOR COMPUTING

XR device manufacturers increasingly employ four or more cameras to improve tracking accuracy [11], which causes the power consumption of I/O between the cameras and System-on-Chip (SoC) to become a non-trivial percentage of total system power. The system-level power analysis of ILLIXR in Chapter 3 corroborates this by showing I/O power to be a significant contributor to total system power. One technique for reducing I/O power is on-

sensor computing, whereby a part or whole of a computation is performed on the sensor itself to reduce communication between the sensor and the SoC.

In Chapter 5, we demonstrate the use of distributed on-sensor computing to optimize the power consumption of head tracking. We choose head tracking for two reasons. First, the system-level analysis of ILLIXR in Chapter 3 showed head tracking to be a significant consumer of overall CPU cycles and, thus, system power. Second, head tracking, typically performed via Visual-Inertial Odometry (VIO), is of paramount importance in XR; the computations that are performed, the contents of the display, and the sounds played back by the speakers are all dependent on it.

We develop a new architecture for VIO that uses distributed on-sensor computing to reduce I/O power [12, 13]. Our design caches parts of the camera images, called image patches, within the image sensors themselves, and uses a centralized directory to coordinate the transfer of image patches between sensors. As a result, it reduces I/O power by 37× and overall VIO subsystem power by 1.3×. Importantly, the general principle of sensor-compute partitioning can be extended to other components of an XR system which possess similar image processing characteristics as VIO, and has since been demonstrated in the context of eye tracking [14] and hand tracking [15]. Furthermore, our design has implications for security and privacy, as it prevents raw images from leaving the image sensors, and for the physical design of the wearable device, as a distributed design may afford more flexibility in terms of sensor and wire placement.

1.5 ONLINE FREQUENCY CONTROL

A second principle for reducing the power consumption of both the sensors and the components driven by them is online frequency control. If a particular component is flexible in terms of the frequency of its execution, this flexibility can be leveraged by a QoE-aware scheduler to run the component as infrequently as possible while maintaining acceptable quality. By operating at a lower frequency and performing less work altogether, significant power savings can be achieved. Furthermore, the frequency of the sensor driving the component can be also reduced, reducing both sensor and I/O power. In Chapter 6, we show that scene reconstruction, a key enabling technology for augmented and mixed reality, is amenable to running at different frequencies, and design an online controller that chooses its operation frequency in real-time, reducing average power by 4.9× compared to a constant frequency baseline [16]. We summarize results for extending online frequency control to other parts of the system in Section 1.7.3.

1.6 SCALABLE HARDWARE MAPPING

Hardware specialization is often necessary in order to achieve milliwatt-scale operation. However, our component analysis from Chapter 4 suggests that it is infeasible to build a unique accelerator for each task of each component. Thus, we explore scalable hardware mapping as a third design principle to optimize system power in Chapter 7. We approach the problem of hardware specialization by carefully balancing the use of existing hardware blocks with the design of new ones. We map one task of scene reconstruction to an Image Signal Processor (ISP), another to a Digital Signal Processor (DSP), and design an accelerator for the third, most expensive task. Combined with online frequency control, our hardware mapping brings scene reconstruction subsystem power in the 10 mW domain, two orders of magnitude lower than an embedded GPU. In Section 1.7.4, we summarize results for a tool that automatically finds acceleration candidates, and in the future aims to extend scalable hardware mapping to other parts of the system.

1.7 RESEARCH PROJECTS USING ILLIXR

Finally, we highlight various projects using or planning on using ILLIXR to demonstrate the impact that ILLIXR has already had on the research community over a short period of time. We collaborated on several of these projects, but they were led by others, and are thus not included as separate chapters in this thesis.

1.7.1 Compute Offload

There are several optimization techniques that simply cannot be explored without the ability to modify the internals of an XR runtime and/or without an end-to-end system. With ILLIXR, exploring these research directions becomes possible. One such technique is offloading compute to an edge or cloud server. Our analysis of the pose estimation pipeline of ILLIXR showed that one (cheap) stage of the pipeline could compensate for latency in another (expensive) stage. Inspired by this finding, we offload the expensive stage to a server on the AWS cloud over WiFi, 4G, and 5G [17]. By doing so, we reduce CPU power by 2.5× and total system power by 1.3×, while maintaining an acceptable QoE. In contrast to accelerating pose estimation in hardware, this approach maintains algorithmic flexibility. Importantly, it is a general design principle, and can be applied to other components of the system as well, such as scene reconstruction and rendering.

1.7.2 QoE-aware Cross-Component Approximate Computing

Another end-to-end design principle that we explore using ILLIXR is QoE-aware cross-component approximate computing, where errors in one component can be ameliorated in another component. We explore the power, performance, and quality tradeoffs involved in implementing eye-tracked foveated rendering [18]. By approximating the eye tracker and compensating for its errors in the foveated renderer, we are able to improve the energy-efficiency of eye tracking by 20× and of foveated rendering by 1.7× compared to baseline implementations, while maintaining QoE.

1.7.3 QoE-aware Scheduling

Extending the idea of online frequency control to the entire end-to-end system, we use ILLIXR to perform QoE-aware scheduling of multiple components together within the runtime. We design a scheduling framework, Catan [19, 20], that takes as input an end-to-end QoE target and a list of components, and both determines their frequencies and allocates CPUs and CPU time to them. Using Catan, we are able to maintain an acceptable QoE when running a subset of ILLIXR on a CPU with just one core.

1.7.4 Automated Accelerator Selection

Similarly, we extend scalable hardware mapping to the entire end-to-end system. In the first step towards that direction, we develop Trireme [21], an automated design space exploration tool that extends AccelSeeker [22] to leverage the parallelism exposed by the HPVM IR [23] to determine which tasks of a component to accelerate. When applied to the audio pipeline of ILLIXR, it generates hardware which improves performance by 12×–18× compared to a software baseline. In the future, Trireme can be extended to automatically identify common compute primitives across tasks and components, as we did manually in Chapter 7.

1.7.5 Wider Adoption

ILLIXR is already starting to see adoption in both academia and industry. In addition to the aforementioned research projects, ILLIXR is being used to design novel accelerator communication interfaces, and has been used for eye-tracked holography [24]. We have also

formed an industry-backed consortium on the basis of ILLIXR.[5] The goal of the consortium is to democratize XR research, development, and benchmarking, and provide a reference research testbed to the community. Companies such as Arm, Cisco, IBM, Intel, Meta, Micron, and NVIDIA are either members of the consortium or are using or funding ILLIXR for research in network technologies for compute offload, SoC design, mixed reality pipelines, and graphics. Finally, ILLIXR has to led to a $ 1M NSF Computer and Information Science and Engineering (CISE) Community Research Infrastructure grant [25], which aims to expand the scope of ILLIXR, and make it a community research infrastructure readily accessible and usable by the wider CISE research community.

[5] `https://illixr.org`

CHAPTER 2: ILLIXR: FIRST OPEN-SOURCE EXTENDED REALITY TESTBED

As we move from the era of general-purpose, homogeneous cores on chip to domain-specific, heterogeneous *system*-on-chip architectures, benchmarks need to follow the same trajectory. While previous benchmarks comprising of suites of independent applications (e.g., Parsec [26], Rodinia [27], SPEC [28, 29], SPLASH [30, 31, 32], and more) sufficed to evaluate general-purpose single- and multicore architectures, there is now a need for a *full-system-benchmark* methodology, better viewed as a *full system testbed*, to design and evaluate system-on-chip architectures. Such a methodology must bring together the diversity of components that will interact with each other in the domain-specific system and also be flexible and extensible to accept future new components.

An XR full-system-benchmark or testbed will continue to enable traditional research for accelerating a given XR component with conventional metrics such as power, performance, and area for that component, but will additionally allow evaluations for the end-to-end impact on the system. More importantly, the integrated system will enable new research that co-designs acceleration of its multiple, diverse, and demanding components across the full stack, driven by end-to-end user experience.

To that end, we develop ILLIXR (Illinois Extended Reality Testbed), the first open-source XR full-system testbed [9] in this chapter. There were two challenges in the development of ILLIXR. First, ILLIXR required expertise in a large number of sub-domains (e.g., robotics, computer vision, graphics, optics, and audio). We consulted with many academic and industry experts in these sub-domains and XR systems. Through these discussions, we identified a representative XR workflow with state-of-the-art algorithms and open source codes for its constituent components, which we integrated into the ILLIXR testbed. ILLIXR has now been vetted through presentations to several XR industry and academic groups and events (e.g., [33]) and is the basis of an industry-backed consortium to enable XR research and standardize XR systems benchmarking [34].

Second, until recently, commercial XR devices had proprietary interfaces. For example, the interface between an Oculus head mounted device (HMD) runtime and the Unity or Unreal game engines that run on the HMD has been closed as are the interfaces between the different components within the HMD runtime. OpenXR [7], an open standard, was released in July 2019 to partly address this problem by providing a standard for XR device runtimes to interface with the applications that run on them. ILLIXR leverages Monado [8], an open implementation of OpenXR, to provide an OpenXR compliant XR system. However, OpenXR is still evolving, it does not address the problem of interfaces between the different

components within an XR system, and there were limited OpenXR compliant applications (e.g., game engines and games) during the development of ILLIXR. Nevertheless, we are able to report results with sophisticated applications running on ILLIXR, and the rapidly growing popularity of OpenXR opens up a rich space of applications that ILLIXR can support to drive future research.

2.1 XR REQUIREMENTS AND METRICS

Table 2.1: Ideal requirements of VR and AR vs. state-of-the-art devices, Varjo VR-3 for VR and Microsoft HoloLens 2 for AR. VR devices are typically larger and so afford more power and thermal headroom. The Varjo VR-3 offloads most of its work to an attached server, so its power and area values are not meaningful. The ideal case requirements for power, area, and weight are based on current devices which are considered close to ideal in these (but not other) respects – Snapdragon 835 [35] in the Oculus Quest VR headset and APQ8009w [36] in the North Focals small AR glasses. The average power budget for consumer AR glasses is approximately 500 mW [37]. Assuming a 400 mW display [38, 39], all the components must run in 100 mW or less.

Metric	**Varjo VR-3 [40]**	**Ideal VR [41, 42]**	**Microsoft HoloLens 2**	**Ideal AR [37, 41, 42]**
Resolution (MPixels)	15.7	200	4.4 [43]	200
Field-of-view (Degrees)	115	Full: 165×175 Stereo: 120×135	52 diagonal [44, 45]	Full: 165×175 Stereo: 120×135
Refresh rate (Hz)	90	90 – 144	120 [46]	90 – 144
Motion-to-photon latency (ms)	< 20	< 20	< 9 [47]	< 5
Power (W)	N/A	1 – 2	> 7 [48, 49, 50]	0.1 – 0.2
Silicon area (mm^2)	N/A	100 – 200	> 173 [48, 51]	< 100
Weight (grams)	944	100 – 200	566 [43]	10s

XR requires devices that are lightweight, mobile, and all day wearable, and provide sufficient compute at low thermals and energy consumption to support a high quality of experience (QoE). Table 2.1 summarizes values for various system-level quality related metrics for current state-of-the-art XR devices and the aspiration for ideal futuristic devices. These requirements are driven by both human anatomy and usage considerations; e.g., ultra-high performance to emulate the real world, extreme thermal constraints due to contact with human skin, and light weight for comfort. Thermal constraints limit instantaneous power

to approximately 1 W [52], and battery constraints limit average power to approximately 500 mW [37].

We identified the values of the various aspirational metrics through an extensive survey of the literature [3, 41, 53]. Although there is no consensus on exact figures, there is an evolving consensus on approximate values that shows orders of magnitude difference between the requirements of future devices and what is implemented today (making the exact values less important for our purpose). For example, the AR power gap alone is two orders of magnitude. Coupled with the other gaps (e.g., two orders of magnitude for resolution), the overall system gap is many orders of magnitude across all metrics.

2.2 THE ILLIXR SYSTEM

Figure 2.1 presents the ILLIXR system. ILLIXR captures components that belong to an XR runtime such as Oculus VR (OVR) and are shipped with an XR headset. Applications, often built using a game engine, are separate from the XR system or runtime, and interface with it using the runtime's API. For example, a game developed on the Unity game engine for an Oculus headset runs on top of OVR, querying information from OVR and submitting rendered frames to it using the OVR API. Analogously, applications interface with ILLIXR using the OpenXR API [7] (we leverage the OpenXR API implementation from Monado [8]). Consequently, ILLIXR does not include components from the application, but captures the performance impact of the application running on ILLIXR. We collectively refer to all application-level tasks such as user input handling, scene simulation, physics, rendering, etc. as "application" in the rest of this thesis.

As shown in Figure 2.1, ILLIXR's workflow consists of three pipelines – perception, visual, and audio – each containing several components and interacting with each other through the ILLIXR communication interface and runtime. Figure 2.2 illustrates these interactions – the left side presents a timeline for an ideal schedule for the different components and the right side shows the dependencies between the different components (enforced by the ILLIXR runtime). We distilled this workflow from multiple sources that describe different parts of a typical VR, AR, or MR pipeline, including conversations with several experts from academia and industry.

ILLIXR represents a state-of-the-art system capable of running typical XR applications and providing an end-to-end XR user experience. Nevertheless, XR is an emerging and evolving domain and no XR experimental testbed or commercial device can be construed as complete in the traditional sense. Compared to specific commercial headsets today, ILLIXR may miss a component (e.g., Oculus Quest 2 provides hand tracking) and/or it may have

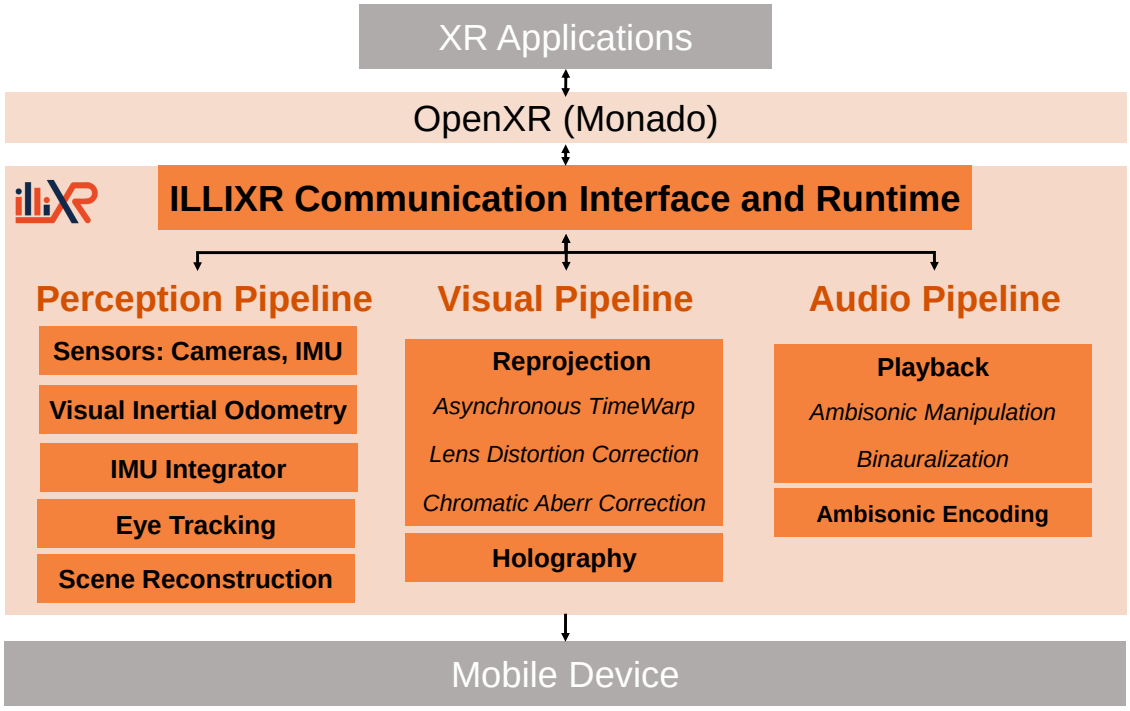


Figure 2.1: The ILLIXR system and its relation to XR applications, the OpenXR interface, and mobile platforms.

additional or more advanced components (e.g., except for HoloLens 2, most current XR systems do not have scene reconstruction). Further, while ILLIXR supports state-of-the-art algorithms for its components, new algorithms are continuously evolving. ILLIXR therefore supports a modular and extensible design that makes it relatively easy to swap and to add new components.

The system presented here assumes all computation is done at the device (i.e., no offloading to the cloud, server, or other edge devices) and we assume a single user (i.e., no communication with multiple XR devices). Thus, the workflow in this chapter does not represent any network communication. We discuss cloud offloading and multiparty XR in Section 9.1 and 9.9, respectively.

Sections 2.3– 2.5 next describe the three ILLIXR pipelines and their components, Section 2.6 describes the modular runtime architecture that integrates these components, and Section 2.7 describes the metrics and telemetry support in ILLIXR. Table 2.2 summarizes the algorithm and implementation information for each component in the three pipelines. ILLIXR already supports multiple, easily interchangeable alternatives for some components; for lack of space, we pick one alternative, indicated by a * in the table, for detailed results in this thesis.

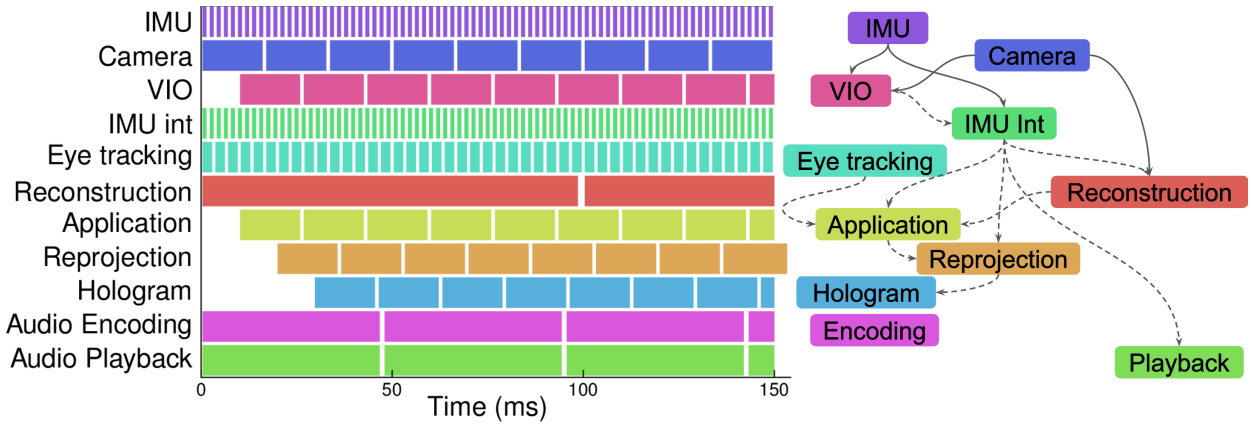


Figure 2.2: Interactions between ILLIXR components. The left part shows an ideal ILLIXR execution schedule and the right part shows inter-component dependencies that the ILLIXR scheduler must maintain (Section 2.6). Solid arrows are synchronous and dashed are asynchronous dependencies.

2.3 PERCEPTION PIPELINE

The perception pipeline translates the user's physical motion into information understandable to the rest of the system so it can render and play the new scene and sound for the user's new position.

2.3.1 Camera

Modern XR systems contain several different types of cameras to drive the perception pipeline, including, but not limited to, stereo, RGB, and depth cameras. ILLIXR can work with any commercial camera. We provide support for ZED Mini, Intel RealSense, and DepthAI cameras. In this thesis, we use a ZED Mini [68] as it provides IMU readings, stereo RGB images, and depth images all together with a simple API. The camera component first asks the camera to obtain new stereo camera images and then copies these images into main memory.

Cameras provide significant information about the user's environment, but run at a relatively low frequency (typically $\leq 30\,\text{Hz}$), and are prone to errors from exposure and lighting changes; e.g., outdoors and in dark environments.

2.3.2 IMU

To address the shortcomings of cameras, XR systems also contain one or more inertial measurement units (IMUs). IMUs consist of an accelerometer and a gyroscope which mea-

Table 2.2: ILLIXR component algorithms and implementations. GLSL stands for OpenGL Shading Language. * represents the implementation alternative for which we provide detailed results.

Component	Algorithm	Implementation
Perception Pipeline		
Camera	ZED SDK* [54]	C++
Camera	DepthAI SDK [55]	C++
Camera	Intel RealSense SDK [56]	C++
IMU	ZED SDK* [54]	C++
IMU	DepthAI SDK [55]	C++
IMU	Intel RealSense SDK [56]	C++
VIO	OpenVINS* [57]	C++
VIO	ORB-SLAM3 [58]	C++
VIO	Kimera-VIO [59]	C++
IMU Integrator	RK4* [57]	C++
IMU Integrator	GTSAM [60]	C++
Eye Tracking	RITnet [61]	C++, CUDA
Scene Reconstruction	InfiniTAM* [62]	C++, CUDA
Scene Reconstruction	KinectFusion [63]	C++, CUDA
Scene Reconstruction	ElasticFusion [64]	C++, CUDA, GLSL
Visual Pipeline		
Reprojection	VP-matrix reprojection w/ pose [65]	C++, GLSL
Lens Distortion	Mesh-based radial distortion [65]	C++, GLSL
Chromatic Aberration	Mesh-based radial distortion [65]	C++, GLSL
Adaptive display	Weighted Gerchberg–Saxton [66]	CUDA
Audio Pipeline		
Audio Encoding	Ambisonic encoding [67]	C++
Audio Playback	Ambisonic manipulation, binauralization [67]	C++

sure the user's linear acceleration and angular velocity, respectively. Even though IMUs provide less information than cameras, they run at a high frequency (typically $\geq 1\,\text{kHz}$) and work in all conditions. Thus, cameras and IMUs work together synergistically by addressing each other's shortcomings.

ILLIXR's IMU component queries the IMU at fixed intervals to obtain linear acceleration from the accelerometer and angular velocity from the gyroscope. In addition to retrieving the IMU values, this component converts the angular velocity from $^\circ\,\text{s}^{-1}$ to $\text{rad}\,\text{s}^{-1}$, and timestamps the readings for use by other components, such as head tracking.

2.3.3 Head Tracking

XR is an inherently egocentric domain, which necessitates knowing the position and orientation of the user's head, referred to as the six degrees of freedom or 6DOF pose, at all times. The pose is typically obtained via Simultaneous Localization and Mapping (SLAM), a technique that can track the user's motion without the use of external markers in the environment.

There are a large number of distinct algorithms that have been proposed for SLAM for a variety of scenarios, including for robots, drones, and XR. The constraints, requirements, and sensor inputs for different scenarios are different, leading to different optimal choices; e.g., some scenarios only require odometry (enabling sparse algorithms) while others require detailed scene reconstruction (favoring dense algorithms). A comprehensive overview of SLAM can be found in the survey by Cadena et al. [69]. We optimize head tracking in Chapter 5.

We support OpenVINS [70], Kimera-VIO [71], and ORB-SLAM3 [72] in ILLIXR. In this thesis, we focus on OpenVINS as it has been shown to be more accurate than other popular algorithms such as VINS-Mono, VINS-Fusion, and OKVIS [70]. OpenVINS uses feature tracking and an MSCKF-based backend [73] to obtain the pose of the user. Both OpenVINS and Kimera-VIO do not have mapping capabilities, and are therefore called Visual-Inertial Odometry (VIO) algorithms, whereas ORB-SLAM3 is a full SLAM algorithm.

2.3.4 IMU Integrator

Head tracking only generates poses at the rate of the camera, which is typically low. To obtain high speed estimates of the user's pose, an IMU integrator integrates incoming IMU samples from the IMU to obtain a relative pose and adds it to the head tracker's latest pose to obtain the current pose of the user. ILLIXR has two IMU integrators available: one utilizes GTSAM to perform integration [74, 75] and the other uses RK4, inspired by OpenVINS' IMU integrator [70]. We use the RK4 integrator in this thesis.

2.3.5 Eye Tracking

Eye or gaze tracking enables several use cases in XR: foveated rendering, automatic display calibration, and using gaze as an input modality. Foveated rendering [18] is a rendering technique which leverages properties of the human visual system to render the the image at full fidelity in the center of the user's gaze, and at lower fidelity in the peripheral vision of the

user to save time and energy. We discuss codesigning eye tracking and foveated rendering in Section 9.3. Automatic display calibration adjusts the properties of the display based on the location of the eyes relative to the display. Finally, gaze data can also be used to provide inputs to the system, such as keystrokes, which is an important aspect of enabling XR for handicapped people.

We use RITnet for eye-tracking [61] as it won the OpenEDS 2019 eye tracking challenge [76] with an accuracy of 95.3%. RITnet is a deep neural network that combines U-Net [77] and DenseNet [78]. The network takes as input a grayscale eye image and segments it into the background, iris, sclera, and pupil. An ellipse fitting algorithm then uses the location of the pupil to compute the center of the pupil and gaze direction.

2.3.6 Scene Reconstruction

Scene reconstruction enables both AR and MR applications by generating a 3D representation of the user's environment. This 3D representation, usually shared with the application as a mesh, can be used to generate realistic lighting, occlusion, and physics for virtual objects.

Scene reconstruction is typically performed by employing dense SLAM algorithms. As with head tracking, there is a plethora of such algorithms, and they differ in how they represent the environment (volume vs. surfaces), and how they organize the map (octrees, hash tables, graphs, etc.). A more detailed survey appears in the work by Zollhöfer et al. [79]. We tackle scene reconstruction in Chapter 6.

ILLIXR supports InfiniTAM [80], KinectFusion [81], and ElasticFusion [82]. In this thesis, we focus on InfiniTAM, which uses a volume-based map representation, and organizes it using a hash table. In each frame, incoming color and depth data from the camera is used to update the relevant parts of the volume in the map.

In general, scene reconstruction algorithms perform their own pose estimation using Iterative Closest Point (ICP) [83]. However, in an XR system, the pose from the head tracker can be reused [84, 85], which eliminates ICP and its associated pre-processing tasks, reducing power and system utilization. Nonetheless, reusing pose in scene reconstruction is a non-trivial task, as head tracking poses are usually not optimized for frame-to-frame consistency, and result in expensive post hoc modifications to the 3D map [86]. Consequently, we take ICP into account in our analysis of scene reconstruction in Chapter 4.

2.4 VISUAL PIPELINE

The visual pipeline takes information about the user's new pose from the perception pipeline, the submitted frame from the application, and produces the final display using two parts: asynchronous reprojection and display.

2.4.1 Asynchronous Reprojection

Asynchronous reprojection corrects the rendered image submitted by the application for optical distortions and compensates for latency from the rendering process (which adds a lag between the user's movements and the delivery of the image to the user). The reprojection component reprojects the latest available frame based on a prediction of the user's movement, resulting in less perceived latency. The prediction is performed by obtaining the latest pose from the IMU integrator and extrapolating it to the display time of the image using a constant acceleration model [87].[6] Reprojection is also used to compensate for frames that might otherwise be dropped because the application could not submit a frame on time to the runtime. Reprojection is critical in XR applications, as any perceived latency causes discomfort [88, 89, 90].

In addition to latency compensation, the reprojection component performs lens distortion correction and chromatic aberration correction. Most XR devices use small displays placed close to the user's eyes; therefore, some optical solution is required to create a comfortable focal distance for the user [91]. These optics usually induce significant optical distortion and chromatic aberration – lines become curved and colors are non-uniformly refracted. These optical affects are corrected by applying reverse distortions to the rendered image, such that the distortion from the optics is mitigated [65].

Our reprojection, lens distortion correction, and chromatic aberration correction implementation is derived from [92]. We extracted the distortion and aberration correction, and reprojection shaders from this reference, and implemented our own version in OpenGL that integrates reprojection, lens distortion correction, and chromatic aberration correction into one component to improve performance. Our implementation only supports rotational reprojection, and is colloquially known as *timewarp*. A more advanced version which also performs positional reprojection, called *spacewarp*, is currently under development and was the subject of an undergraduate thesis [93].

Recently, the purview of asynchronous reprojection has expanded significantly to include

[6] Although pose prediction can theoretically reduce pose accuracy, the prediction is typically performed over a time horizon of a few milliseconds, over which the constant acceleration model works well and does not deviate from the actual pose of the user.

other forms of image post-processing. Notably, NVIDIA DLSS [94] and AMD FSR [95] are two popular commercial solutions that improve both image quality and application frame rates by performing image upscaling, denoising, anti-aliasing, and motion vector-based interpolation. AMD FSR is open-source, and a viable candidate for incorporation into the visual pipeline of ILLIXR.

2.4.2 Display

The fixed focal length of modern display optics causes a vergence-accommodation conflict (VAC), where the user's eyes converge at one depth, which varies with the scene, and the eye lenses focus on another, which is fixed due to the display optics. VAC is a common cause of headaches and fatigue in modern XR devices [96, 97, 98, 99]. One possible solution to this problem is computational holography, wherein a phase change is applied to each pixel using a Spatial Light Modulator [100] to generate several focal points instead of just one. Since humans cannot perceive depths less than 0.6 diopters apart [101], typically 10 or so depth planes are sufficient. The per-pixel phase mask is called a hologram. For a comprehensive discussion of alternatives to computational holography, see Matsuda et al. [102] and Koulieris et al. [103].

In ILLIXR, we use Weighted Gerchberg–Saxton (GSW) [104] to generate holograms. We used a reference CUDA implementation of GSW [66], and extended it to support RGB holograms of arbitrary size. Recently, neural approaches [105, 106, 107] have begun to outperform GSW in terms of both speed and visual fidelity, and can be incorporated into ILLIXR as the technology matures.

2.5 AUDIO PIPELINE

Spatial audio is audio that captures the positional information of a sound source in addition to its amplitude. Spatial audio is a critical component of XR as it is imperative that the user perceive virtual sounds as real sounds, which can only be achieved via sound source localization; i.e., by using the source's position [108].

One popular way of capturing spatial audio is Higher Order Ambisonics (HOA) [109]. HOA allows sound sources to be encoded into a spherical soundfield which can then be manipulated and played back independently. The order of HOA determines how many virtual channels exist in the generated soundfield. Specifically, $channels = (order + 1)^2$. For instance, order 3 HOA results in 16 virtual channels. This is an important property of HOA: the number of virtual channels in the soundfield is only dependent on the order and not on

the physical number of sources that have been encoded into the field. As a result, HOA is a popular format for XR, as it can encode tens or even hundreds of sound sources without any increase in soundfield complexity [110].

ILLIXR's audio pipeline is responsible for generating realistic spatial audio and is composed of audio encoding and playback. The two use cases are typically mutually exclusive: applications typically have pre-encoded HOA audio and only require playback; recording in, say, a studio session only involves encoding a certain (perhaps large) number of sound sources. While there are cases where both are required, such as teleconferencing, the major component in those scenarios is playback. This is due to the fact that encoding a single sound source (i.e., the user's voice) consumes negligible time compared to playback. Thus, recording and playback can be safely modelled separately. Our audio pipeline uses `libspatialaudio`, an open-source HOA library [67].

2.5.1 Encoding

We perform spatial audio encoding by encoding several different monophonic sound sources simultaneously into an HOA soundfield [111]. The Ambisonic encoder takes into account both the direction of the source and the attenuation and delay caused by the distance of the source. It then calculates the contribution of the source to each of the virtual sound channels in the soundfield. The final soundfield is obtained by simply adding the individual soundfields.

2.5.2 Playback

Playback is a two-step process. First, the encoded soundfield is manipulated using head tracking information. Based on the user's head rotation and movement, the soundfield is rotated and zoomed, respectively. This step includes computing transformation matrices and applying them to each virtual channel of the soundfield [112]. Optionally, a psychoacoustic filter can be applied to the soundfield before performing the rotations in order to improve the quality of the output [109, 113, 114].

Once the desired soundfield has been obtained, binauralization [115] maps it to physical speakers, typically headphones in XR devices (HOA can decode to any number of speakers; headphones are the most common speaker configuration in XR). This involves summing the contribution of each virtual channel to the left and right speakers. The most salient aspect of binauralization is the application of HRTFs or Head-Related Transfer Functions.

HRTFs are digital filters which capture how incoming sound is modified by a user's nose,

head, and outer ear shape in real life. The binauralization process applies HRTFs (implemented as frequency-domain convolutions) to the decoded sound in order to emulate these real-life effects. Adding such cues allows the digitally rendered sound to mimic real sound, helping the brain localize sounds akin to real life. There are separate HRTFs for the left and right ears due to the asymmetrical shape of the human head.

2.6 RUNTIME

Figure 2.2 shows an ideal execution timeline for the different XR components and their temporal dependencies. On the left, each colored rectangle boundary represents the period of the respective component — ideally, the component would finish execution before its next invocation. The right side shows a static representation of the dependencies among the components, illustrating the interaction between the different pipelines.[7] We say a consumer component exhibits a *synchronous* dependence on a producer component if the former has to wait for the last invocation of the latter to complete (solid arrow). An *asynchronous* dependence is softer, where the consumer can start execution with data from a previously complete invocation of the producer component (dashed arrow).

An XR system is unlikely to follow the idealized schedule shown in Figure 2.2 (left) due to shared and constrained resources and variable running times. Thus, an explicit runtime is needed for effective resource management and scheduling while maintaining the inter-component dependencies, resource constraints, and quality of experience. The ILLIXR runtime schedules resources while enforcing dependencies among the components, in part deferring to the Linux kernel and GPU driver. To achieve extensibility, ILLIXR is divided into 1) a communication framework, 2) a set of plugins, and 3) a plugin-loader.

The communication framework is structured around *event streams*. Event streams support writes, asynchronous reads, and synchronous reads. Synchronous reads allow a consumer to see every value produced by the producer, while asynchronous reads allow a consumer to ask for the latest value.

Plugins are distributed as shared-object files for the runtime to load. The runtime gives the plugins access to other plugins but in a limited sense; they can only interact through event streams. This architecture allows for modularity. Each component in Table 2.2 is implemented in its own plugin.

The plugin loader can load a list of plugins defined at runtime. Each plugin is interchangeable with another as long as it complies with the event-stream interface and writes to the

[7] An additional constraint is that reprojection should be scheduled as late as possible (before the next vsync) so it has the freshest pose to generate the reprojected frame to be finally displayed.

same event streams in the same manner. Future researchers can test alternative implementations of a single plugin without needing to reinvent the rest of the system. Development can iterate quickly, because plugins are compiled independently and linked dynamically.

ILLIXR is supported as a device driver in the Monado OpenXR runtime [8]. This allows ILLIXR to run OpenXR applications, including those developed using game engines such as Unity, Unreal, and Godot (currently only Godot and Unreal have OpenXR support on Linux).

Finally, although ILLIXR supports live sensor inputs (e.g., through cameras and IMUs) and an HMD display, it does not *require* such external hardware. To enable universal use and ease experimentation, the only requirement is a computer (desktop/laptop/embedded board). Offline, pre-recorded datasets can be fed to all parts of ILLIXR due to its well-defined and modular communication interfaces. As an example, ILLIXR's offline camera+IMU component reads from a pre-recorded dataset and publishes to the same output stream as a live camera+IMU component, appearing indistinguishable from a real camera/IMU to the rest of the system. Similarly, ILLIXR does not require an HMD and can display the final stereoscopic images to a regular monitor.

2.7 METRICS

ILLIXR provides several metrics to evaluate the goodness of the system. In addition to reporting conventional performance metrics such as per-component frame rate, per-component execution time, and power-related metrics, ILLIXR reports several QoE metrics: 1) motion-to-photon latency [88], a standard measure of the lag between user motion and image updates; 2) Structural Similarity Index Measure (SSIM) [116], one of the most commonly used image quality metrics in XR studies [117, 118, 119]; 3) FLIP [120], a recently introduced image quality metric that addresses shortcomings of SSIM; and 4) pose error [121], a measure of the accuracy of the poses used to render the displayed images.

While ILLIXR currently implements SSIM and FLIP, its pose and image collection infrastructure is generic and extensible, enabling evaluation of other (evolving) metrics for image or video quality. This is important as such metrics for XR are still an active area of research. Notably, both SSIM and FLIP are image metrics, whereas the final output of the visual pipeline is a video, requiring consideration of aspects such as temporal coherence and smoothness (jitter) as well. For instance, VMAF [122], Video ATLAS [123], and FoVVideoVDP [124] have made important steps in this direction.

ILLIXR does not yet compute a quality metric for audio beyond bitrate, but one can be added, such as the recently developed AMBIQUAL [125].

CHAPTER 3: XR FULL-SYSTEM ANALYSIS

In this chapter, we analyze the performance (execution time and throughput), power, and QoE of ILLIXR at a system level. We do so to understand the characteristics of future XR workloads and the implications of these workloads for future XR systems. We quantitatively show that (1) XR presents a rich opportunity for computer architecture and systems research in domain-specific edge systems; (2) fully exploiting this opportunity requires an end-to-end system that models the complex, interacting pipelines in an XR workflow; and (3) ILLIXR is a unique testbed that provides such an end-to-end system, enabling new research directions in domain-specific edge systems architecture in general and for XR in particular. To the best of our knowledge, this is the first publicly available characterization of a full XR system.

3.1 EXPERIMENTAL METHODOLOGY

3.1.1 Experimental Setup

There are no existing systems that meet all the aspirational performance, power, and quality criteria for a fully mobile XR experience as summarized in Section 2.1. For our characterization, we choose to run ILLIXR on two hardware platforms, with three total configurations, representing a broad spectrum of power-performance tradeoffs and current XR devices.

Desktop Platform We use a state-of-the-art desktop system with an Intel Xeon E-2236 CPU (6C12T) and a discrete NVIDIA RTX 2080 GPU. This platform's thermal design power (TDP) rating is far above what is deemed acceptable for a mobile XR system, but it is representative of the platforms on which current tethered systems run (e.g., Varjo VR-3 as in Section 2.1) and it can be viewed as an approximate upper bound for performance of CPU+GPU based near-future embedded (mobile) systems.

Embedded Platform We use an NVIDIA Jetson AGX Xavier development board [126] consisting of an Arm CPU (8C8T) and an NVIDIA Volta GPU. All experiments are run with the Jetson in 10 Watt mode, the lowest possible preset. We use two different configurations – a high performance one (*Jetson-HP*) and a low power one (*Jetson-LP*). We use the maximum available clock frequencies for Jetson-HP and half those for Jetson-LP. These configurations approximate the hardware and/or the TDP rating of several commercial mobile XR devices.

For example, Magic Leap One [127] uses a Jetson TX2 [128], which is similar in design and TDP to our Xavier configuration. HoloLens 2 [48, 49, 50] and the Qualcomm Snapdragon 835 [129] used in Oculus Quest [130] have TDPs in the same range as our two Jetson configurations.

I/O Setup For the live end-to-end ILLIXR system experiments, our I/O setup is as follows. For the perception pipeline, we connect a ZED Mini camera [68] to the above platforms via a USB-C cable. A user walks in our lab with the camera, providing live camera and IMU input to ILLIXR. For the visual pipeline, we run representative VR and AR applications on a game engine on ILLIXR (Section 3.1.3) – these applications interact with the perception pipeline to provide the visual pipeline with the image frames to display.

ILLIXR can display the (corrected and reprojected) images on both a desktop LCD monitor and a North Star AR headset [131] connected to the above hardware platforms. In this chapter, we use a desktop monitor due to its ability to provide multiple resolution levels and refresh rates for experimentation. Although this means that the user does not see the display while walking with the camera, we practice a trajectory that provides a reasonable response to the displayed images and use that (live) trajectory to collect results. Finally, for the audio pipeline, we use pre-recorded input. We run our experiments for approximately 30 seconds each.

3.1.2 Integrated ILLIXR System Configuration

The end-to-end integrated ILLIXR configuration in our experiments uses the components as described in Table 2.2 except for scene reconstruction, eye tracking, and hologram. The OpenXR standard only recently added interfaces for scene reconstruction and eye tracking. We, therefore, do not have any applications available to use these components in an integrated setting. Although we can generate holograms, we do not yet have a holographic display (Section 2.4.2). We do report results in standalone mode for these components using off-the-shelf component-specific datasets in Chapter 4.

Configuring an XR system requires tuning multiple parameters of the different components to provide the best end-to-end user experience on the deployed hardware. This is a complex process involving the simultaneous optimization of many QoE metrics and system parameters. Currently tuning such parameters is a manual, mostly ad hoc process. Table 3.1 summarizes the key parameters for ILLIXR that required system-level tuning, the range available in our system for these parameters, and the final value we chose at the end of our manual tuning. We make initial guesses based on our intuition, and then choose final

Table 3.1: Key ILLIXR parameters that required manual system-level tuning. Several other parameters were tuned at the component level. The first column lists the primary component that was tuned along with dependent components in parentheses.

Component	**Parameter Range**	**Tuned**	**Deadline**
Camera (VIO)	Frame rate = 15 – 100 Hz	15 Hz	66.7 ms
	Resolution = VGA – 2K	VGA	–
	Exposure = 0.2 – 20 ms	1 ms	–
IMU (Integrator)	Frame rate = $\leq$800 Hz	500 Hz	2 ms
Display (Visual pipeline, Application)	Frame rate = 30 – 144 Hz	120 Hz	8.33 ms
	Resolution = $\leq$2K	2K	–
	Field-of-view = $\leq$ 180°	90°	–
Audio (Encoding, Playback)	Frame rate = 48 – 96 Hz	48 Hz	20.8 ms
	Block size = 256 – 2048	1024	–

values based on both our perception of the smoothness of the system and profiling results. We expect the availability of ILLIXR will enable new research in more systematic techniques for performing such end-to-end system optimization.

3.1.3 Applications

To evaluate ILLIXR, we use four different XR applications: Sponza [132], Materials [133], Platformer [134], and a custom AR demo application with sparse graphics.

Sponza places the user inside the atrium of the famous Sponza Palace in Dubrovnik, Croatia. The objective of the application is to showcase rendering of high polygon count meshes and global illumination. Materials presents the user with several sphere like objects that are composed of different complex materials, and showcases a variety of physically based rendering (PBR) techniques, such as displacement mapping, subsurface scattering, and anisotropic reflections. Platformer contains a maze with crab-like "enemies" that the user can shoot, and focuses on demonstrating physics and collisions. The AR application contains a single light source, a few virtual objects, and an animated ball, and showcases the overlaying of both stationary and moving virtual objects on the physical world. We develop our own AR application because existing applications in the XR ecosystem all predominantly target VR (outside of Microsoft HoloLens).

The applications were chosen for diversity of rendering complexity, with Sponza being the most graphics-intensive and AR demo being the least. We do not evaluate an MR application as OpenXR has only recently introduced a scene reconstruction interface.

Since Unity and Unreal did not have OpenXR support on Linux when this work was done,

all four applications use the Godot game engine [135], a popular open-source alternative with Linux OpenXR support.

3.1.4 Metrics

Execution Time We use NSight Systems [136] to obtain the overall execution timeline of the components and ILLIXR, including serial CPU, parallel CPU, GPU compute, and GPU graphics phases. VTune [137] (desktop only) and `perf` [138] both provide CPU hotspot analysis and hardware performance counter information. NSight Compute [139] and Graphics [140] provide detailed information on CUDA kernels and GLSL shaders, respectively. We also develop a logging framework that allows ILLIXR to easily collect the wall clock time and CPU time of each of its components with negligible overhead. We determine the contribution of a given component towards CPU time by computing the total CPU cycles consumed by that component as a fraction of the cycles used by all components.

Power and Energy For the desktop, we measure CPU power using `perf` and GPU power using `nvidia-smi` [141] (a standard method to measure power on NVIDIA GPUs [142, 143, 144]). On the embedded platform, we collect power and energy using a custom profiler, similar to [145], that monitors different power rails present on the board [146] to calculate both the average power and average energy for different components of the system: CPU, GPU, DDR, SoC (on-chip microcontrollers; excludes CPU and GPU power), and Sys (display, storage, I/O) [147].

Motion-To-Photon Latency (MTP) We compute motion-to-photon latency as the age of the reprojected image's pose when the frame is submitted for display. This latency is the sum of how old the IMU sample used for pose calculation is,[8] the time taken by reprojection itself, and the wait time until the frame buffer is accepted for display. Mathematically, this can be formulated as: $latency = t_{imu\_age} + t_{reprojection} + t_{swap}$. We do not include $t_{display}$, the time taken to display the frame on screen, in this calculation. This calculation is performed and logged by the reprojection component every time it runs. If reprojection misses vsync, the additional latency is captured in t_{swap}.

Image Quality To compute image quality, we compare the outputs of the actual XR system being studied to those of an idealized configuration that directly receives ground truth

[8]This does not account for pose prediction as is common practice. Pose prediction potentially reduces MTP but it is hard to account for mispredictions.

poses from a dataset (we use Vicon Room 1 Medium [148]). A key implementation challenge was that collecting the post-reprojection images from the GPU for comparison incurs too much overhead and perturbs the run. This is mitigated by collecting images produced by the application renderer (which are cheaper to collect) and the poses generated (ground truth for the idealized system), and then applying reprojection offline (using the above poses) to get the final image that would have been displayed. We compare the reprojected images of the actual and idealized systems to compute both SSIM and FLIP. We report 1-FLIP to be consistent with SSIM, with 0 being no similarity and 1 being identical images.

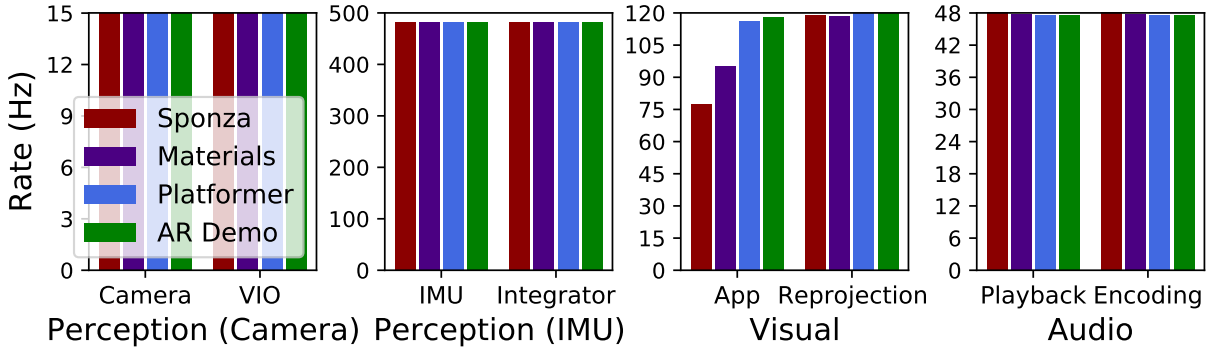
3.2 PERFORMANCE

ILLIXR consists of multiple interacting streaming pipelines and there is no single number that can capture the entirety of its performance (execution time or throughput). We present the performance results for ILLIXR in terms of achieved frame rate and per-frame execution time (mean and standard deviation) for each component, and compare them against the target frame rate and corresponding deadline respectively (Table 3.1). Further, to understand the importance of each component to execution time resources, we also present the contribution of the different components to the total CPU execution cycles.

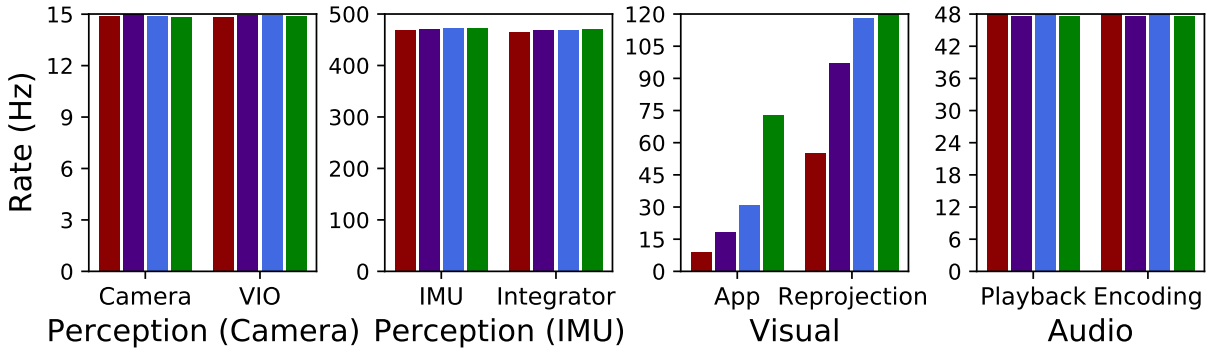
Component Frame Rates Figures 3.1(a)-(c) show each component's average frame rate for each application, for a given hardware configuration. Components with the same target frame rate are presented in the same graph, with the maximum value on the y-axis representing this target frame rate. Thus, we show separate graphs for components in the perception, visual, and audio pipelines, with the perception pipeline further divided into two graphs representing the camera and IMU driven components.

Focusing on the desktop, Figure 3.1a shows that virtually all components meet, or almost meet, their target frame rates (the application component for Sponza and Materials are the only exceptions). The IMU components are slightly lower than the target frame rate only because of scheduling non-determinism at the 2 ms period required by these components. This high performance, however, comes with a significant power cost.

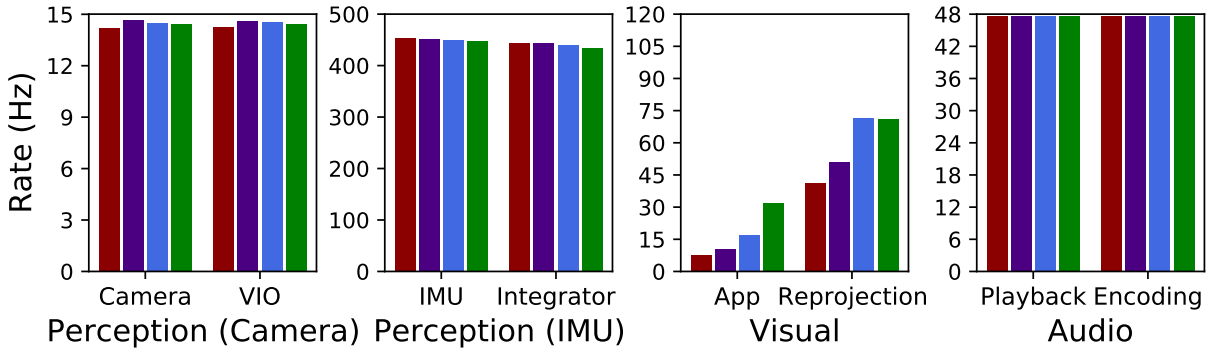
Moving to the lower power Jetson, we find more components missing their target frame rates. With Jetson-LP, only the audio pipeline is able to meet its target. The visual pipeline components – application and reprojection – are both severely degraded in all cases for Jetson-LP and most cases for Jetson-HP, sometimes by more than an order of magnitude. (Recall that the application and reprojection missing vsync results in the loss of an entire



(a) Desktop



(b) Jetson-HP

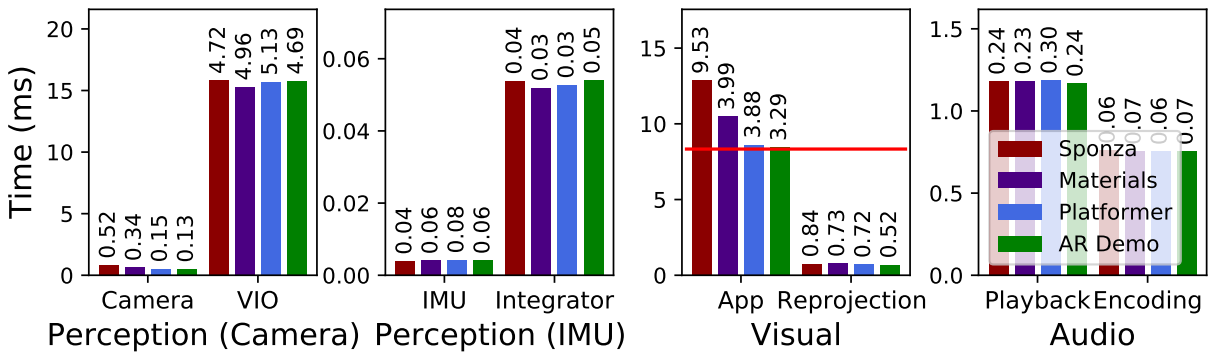


(c) Jetson-LP

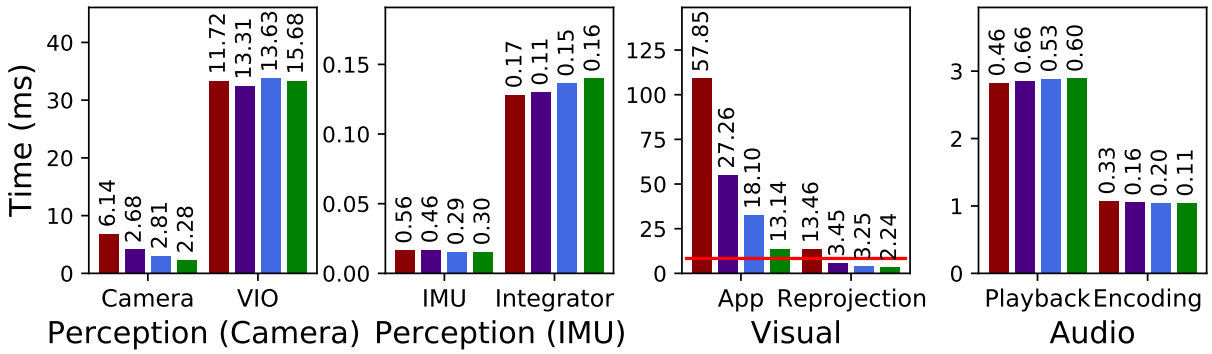
Figure 3.1: Average frame rate for each component in the different pipelines on each application and hardware platform. The y-axis is capped at the target frame rate of the pipeline.

frame. In contrast, other components can catch up on the next frame even after missing the deadline on the current frame, exhibiting higher effective frame rates.)

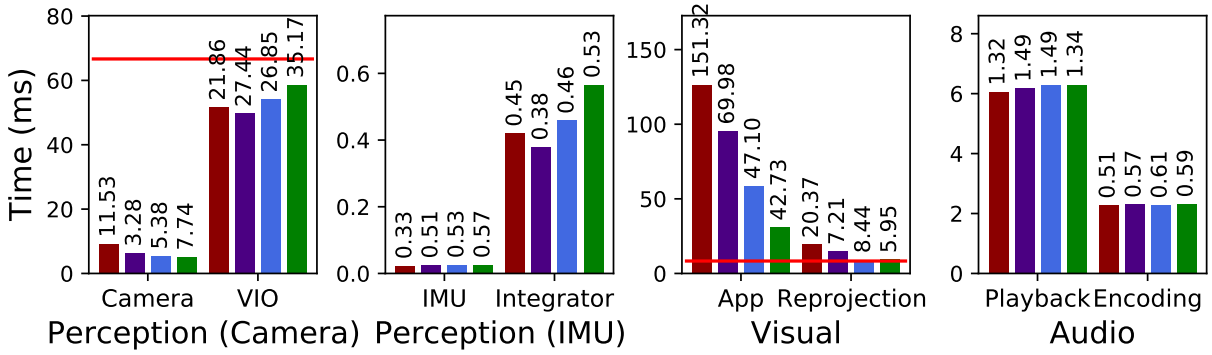
Although we assume modern display resolutions and refresh rates, future systems will support larger and faster displays with larger field-of-view and will integrate more components,



(a) Desktop



(b) Jetson-HP



(c) Jetson-LP

Figure 3.2: Average per-frame execution time for each component for each application and hardware platform. The number on top of each bar is the standard deviation across all frames. The red horizontal line shows the maximum execution time (deadline) to meet the target frame rate. The line is not visible in graphs where the achieved time is significantly below the deadline.

further stressing the entire system.

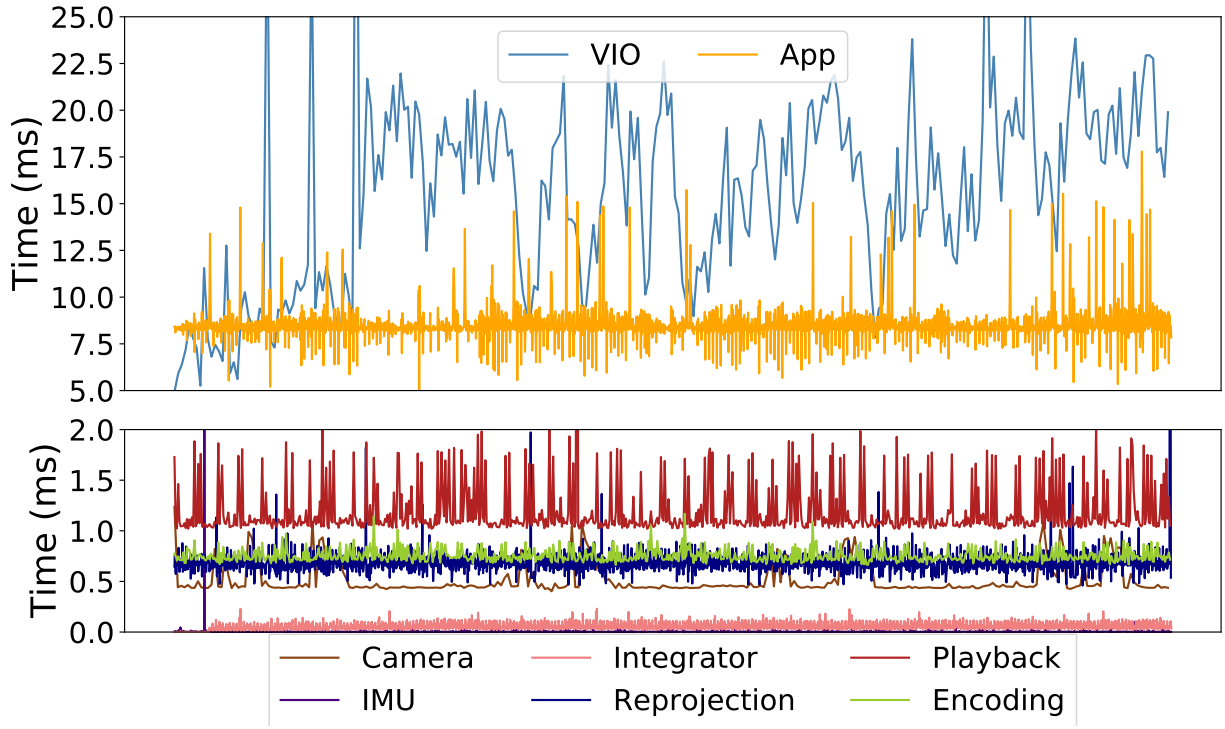


Figure 3.3: Per-frame execution times for Platformer on desktop. The top graph shows head tracking and the application, and the bottom graph shows the remaining components. Note the different scales of the y-axes.

Execution Time per Frame Figures 3.2(a)-(c) show the average execution time (wall clock time) per frame for a given component, organized similar to Figure 3.1. The horizontal line on each graph shows the target execution time or deadline, which is the reciprocal of the target frame rate. Note that the achieved frame rate in Figure 3.1 cannot exceed the target frame rate as it is controlled by the runtime. The execution time per frame, however, can be arbitrarily low or high (because we don't preempt an execution that misses its deadline). The number at the top of each bar shows the standard deviation across all frames.

The mean execution times follow trends similar to those for frame rates. The standard deviations for execution time are surprisingly significant in many cases. For a more detailed view, Figure 3.3 shows the execution time of each frame of each ILLIXR component during the execution of Platformer on the desktop (other timelines exhibit similar behavior). The components are split across two graphs for clarity. We expect variability in head tracking (blue) and application (yellow) since these computations are known to be input-dependent. However, we see significant variability in the other components as well. This variability is due to scheduling and resource contention, motivating the need for better resource allocation, partitioning, and scheduling. A consequence of the variability is that although the mean

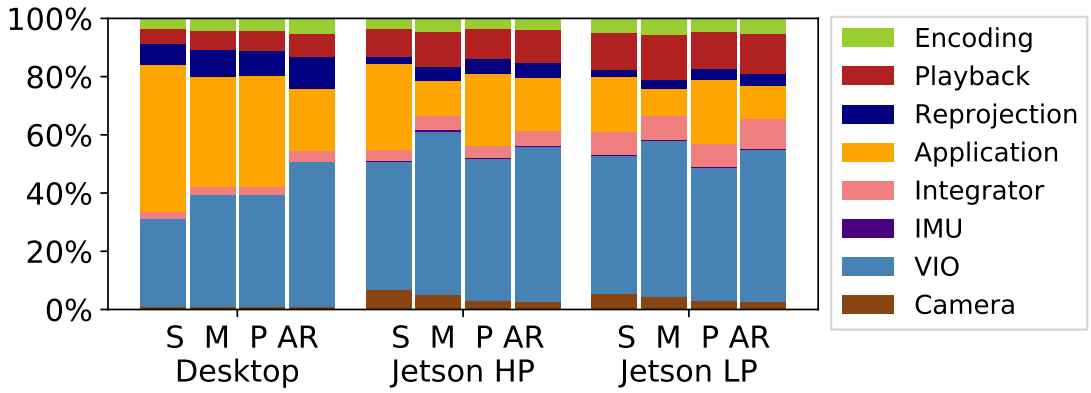


Figure 3.4: Contributions of ILLIXR components to CPU time.

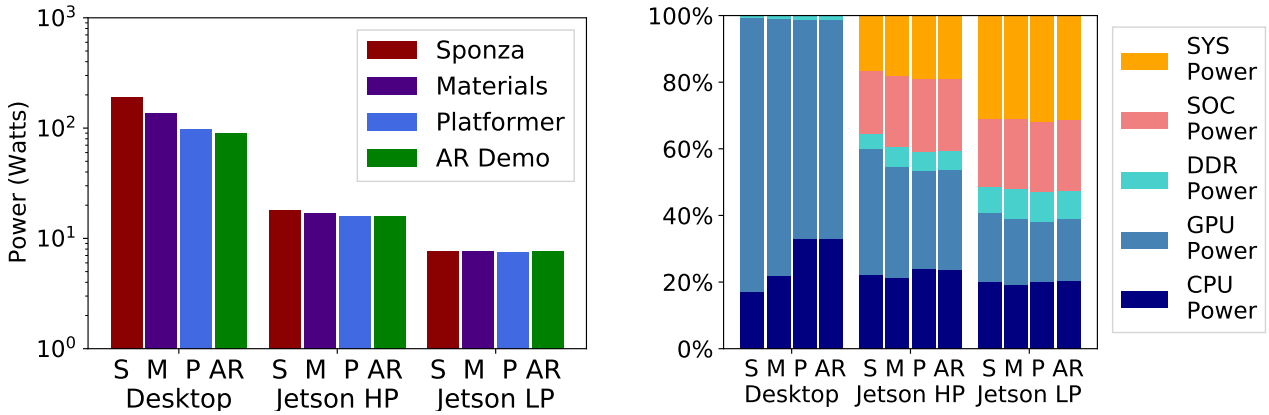
per-frame execution time for some components is comfortably within the target deadline, several frames do miss their deadlines; e.g., head tracking in Jetson-LP. Whether this affects the user's QoE depends on how well the rest of the system compensates for these missed deadlines. For example, even if head tracking runs a little behind the camera, the IMU part of the perception pipeline may be able to compensate. Similarly, if the application misses some deadlines, reprojection may be able to compensate. Section 3.4 provides results for system-level QoE metrics that address these questions.

Distribution of Cycles Figure 3.4 shows the relative attribution of the total cycles consumed in the CPU to the different ILLIXR components for the different applications and hardware platforms. These figures do not indicate wall clock time since one elapsed cycle with two busy concurrent threads will contribute two cycles in these figures. Thus, these figures indicate total CPU resources used. We do not show GPU cycle distribution because the GPU timers significantly perturbed the rest of the execution – most of the GPU cycles are consumed by the application with the rest used by reprojection.

Focusing first on the desktop, Figure 3.4 shows that head tracking and the application are the largest contributors to CPU cycles, with one or the other dominating, depending on the application. Reprojection and audio playback follow next, becoming larger relative contributors as the application complexity reduces. Although reprojection does not exceed 10% of the total cycles, it is a dominant contributor to motion-to-photon latency and, as discussed below, cannot be neglected for optimization.

Jetson-HP and Jetson-LP show similar trends except that we find that the application's and reprojection's relative contribution decreases while other components such as the IMU integrator become more significant relative to the desktop. This phenomenon occurs because with the more resource-contrained Jetson configurations, the application and reprojection

often miss their deadline and are forced to skip the next frame. Thus, the overall work performed by these components reduces, but shows up as poorer end-to-end QoE metrics discussed later (Section 3.4).



(a) Total Power (W)

(b) Power Breakdown

Figure 3.5: (a) Total power (note log scale) and (b) relative contribution to power by different hardware units for each application and hardware platform.

3.3 POWER

Total Power Figure 3.5a shows the total power consumed by ILLIXR running each application on each hardware platform. The power gap from the ideal (Section 2.1) is severe on all three platforms. As shown in Figure 3.5a, Jetson-LP, the lowest power platform, is two orders of magnitude off in terms of the ideal power while the desktop is off by three. As with performance, larger resolutions, frame rates, field-of-views, and more components would further widen this gap.

Power Breakdown Figure 3.5b shows the relative contribution of different hardware units to the total power, broken down as CPU, GPU, DDR, SoC, and Sys. These results clearly highlight the need for studying all aspects of the system. Although the GPU dominates power on the desktop, that is not the case on Jetson. SoC and Sys power are often ignored, but doing so does not capture the true power consumption of XR workloads, which typically exercise all the aforementioned hardware units – SoC and Sys consume more than 50% of total power on Jetson-LP. Thus, reducing the power gap for XR components would require optimizing system-level hardware components as well.

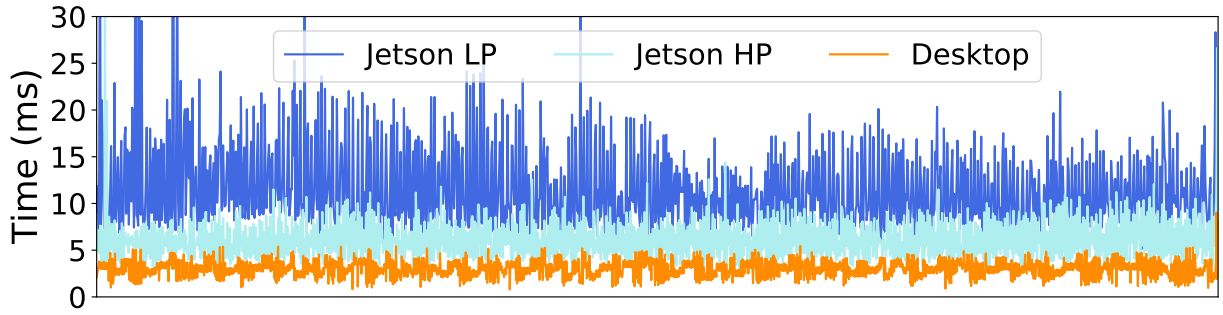


Figure 3.6: Motion-to-photon latency per frame of Platformer.

Quality of an XR experience is often determined through user studies [149, 150, 151]; however, these can be expensive, time consuming, and subjective. ILLIXR, therefore, provides several quantitative metrics to measure QoE. We report both results of visual examination and quantitative metrics below.

Visual Examination A detailed user study is outside the scope of this work, but there were several artifacts in the displayed image that were clearly visible. As indicated by the performance metrics, the desktop displayed smooth images for all four applications. Jetson-HP showed perceptibly increased judder for Sponza. Jetson-LP showed dramatic pose drift and clearly unacceptable images for Sponza and Materials, though it was somewhat acceptable for the less intense Platformer and AR Demo. As discussed in Section 3.2, the average head tracking frame rate for Jetson-LP stayed high, but the variability in the per-frame execution time resulted in many missed deadlines, which could not be fully compensated by the IMU or reprojection. These effects are quantified with the metrics below.

Motion-To-Photon Latency (MTP) Table 3.2 shows the mean and standard deviation of MTP for all cases. Figure 3.6 shows MTP for each frame over the execution of Platformer on all hardware.

Table 3.2 and our detailed per-frame data shows that the desktop can achieve the target VR MTP for virtually all frames. For AR, most cases appear to meet the target, but adding display refresh latency (4.2 ms) significantly exceeds the target. Both Jetsons cannot make the target AR MTP for an average frame. Jetson-HP is able to make the VR target MTP for the average frame for all applications and for most frames for all except Sponza. Jetson-LP shows a significant MTP degradation – on average, it still meets the target VR MTP, but both the mean and variability increase with increasing complexity of the application until

Table 3.2: Motion-to-photon latency in milliseconds (mean±std dev). Target is 20 ms for VR and 5 ms for AR (Table 2.1).

Application	Desktop	Jetson-hp	Jetson-lp
Sponza	3.1 ± 1.1	13.5 ± 10.7	19.3 ± 14.5
Materials	3.1 ± 1.0	7.7 ± 2.7	16.4 ± 4.9
Platformer	3.0 ± 0.9	6.0 ± 1.9	11.3 ± 4.7
AR Demo	3.0 ± 0.9	5.6 ± 1.4	12.0 ± 3.4

Sponza is practically unusable (19.3 ms average with 14.5 ms standard deviation). Thus, the MTP data collected by ILLIXR is consistent with the visual observations reported above.

Table 3.3: Image and pose quality metrics (mean±std dev) for Sponza.

Platform	SSIM	1-FLIP	ATE/degree	ATE/meters
Desktop	0.83 ± 0.04	0.86 ± 0.05	8.6 ± 6.2	0.33 ± 0.15
Jetson-hp	0.80 ± 0.05	0.85 ± 0.05	18 ± 13	0.70 ± 0.33
Jetson-lp	0.68 ± 0.09	0.65 ± 0.17	138 ± 26	13 ± 10

Image and Pose Quality Table 3.3 shows the mean and standard deviation for SSIM, 1-FLIP, ATE/degree, and ATE/distance (Section 2.7) for Sponza on all hardware configurations. We find that all metrics degrade as the hardware platform becomes more constrained. While the trajectory errors reported are clearly consistent with the visual experience, the SSIM and FLIP values seem deceptively high for the Jetsons (specifically the Jetson-LP where head tracking shows a dramatic drift). Quantifying image quality for XR is known to be challenging [152, 153]. While some work has proposed the use of more conventional graphics-inspired metrics such as SSIM and FLIP, this is still a topic of ongoing research [124, 154, 155]. ILLIXR is able to capture the proposed metrics, but our work also motivates and enables research on better image quality metrics for XR experiences.

3.5 KEY IMPLICATIONS FOR ARCHITECTS

Architects have embraced specialization but most research focuses on accelerators for single programs. ILLIXR is motivated by research for specializing an entire domain-specific system, specifically edge systems with constrained resources, high computation demands, and goodness metrics based on end-to-end domain-specific quality of output; e.g., AR glasses, robots/drones, autonomous vehicles, etc. There is emerging consensus that such systems will require quality-driven, end-to-end co-design [2]. Our results characterize end-to-end

performance, power, and QoE of an XR device, exposing new systems research opportunities and demonstrating ILLIXR as a unique testbed to enable exploration of these domain-specific systems, as follows.

Performance, Power, and QoE Gaps Our results quantitatively show that collectively there is several orders of magnitude performance, power, and QoE gap between current representative desktop and embedded class systems and the goals in Section 2.1. The above gap will be further exacerbated with higher fidelity displays and the addition of more components for a more feature-rich XR experience (e.g., scene reconstruction, eye tracking, hand tracking, and holography). We address scene reconstruction in Chapter 6.

While the presence of these gaps itself is not a surprise, we provide the first such quantification and analysis. This provides insights for directions for architecture and systems research (below) as well as demonstrates ILLIXR as a one-of-a-kind testbed that can enable such research.

No Dominant Component Our more detailed results show that there is no one component that dominates all the metrics of interest. While the application and head tracking seem to dominate CPU cycles, reprojection latency is critical to MTP, and audio playback takes similar or more cycles as reprojection. Since image quality relies on accurate poses, frequent operation of the IMU integrator is essential to potentially compensate for missed head tracking deadlines. As mentioned earlier, Hologram dominated the GPU and was not even included in the integrated configuration studied since it precluded meaningful results on the Jetson-LP. Thus, to close the aforementioned gaps, *all* components have to be considered together, even those that may appear relatively inexpensive at first glance. Moreover, we expect this diversity of important components to only increase as more components are included for more feature rich experiences. These observations coupled with the large performance-power gaps above indicate a rich space for hardware specialization research, including research in automated tool flows to determine what to accelerate, how to accelerate these complex and diverse codes, and how to best exploit accelerator-level parallelism [156].

Full-System Power Contributions Our results show that addressing the power gap requires considering *system-level* hardware components, such as display and other I/O, including numerous sensors. While we do not measure individual sensor power, it is included in the Sys power on the Jetson, which is significant. This motivates research in unconventional architecture paradigms such as on-sensor computing to save I/O power; e.g., the image processing tasks of head tracking can be moved to the sensor so that only detected

features and not entire camera frames are sent from the camera to the SoC. We explore this idea further in Chapter 5. To reduce sensor power, sensor parameters can be tuned; e.g., reducing camera exposure can save power at the cost of a darker image. However, sensors are typically shared among components [157], and thus decisions regarding which tasks to move on-sensor and how to dynamically alter exposure must consider the entire system and not just one component. ILLIXR enables this.

Variability and Scheduling Our results show large variability in per-frame processing times in many cases, either due to inherent input-dependent nature of the component (e.g., head tracking) or due to resource contention from other components. This variability poses challenges to, and motivates research directions in, *scheduling, resource partitioning, and allocation of hardware resources.* As presented in Table 3.1, there are a multitude of parameters that need to be tuned for optimal performance of the system. This chapter performs manual tuning to fix these parameters; however, ideally they would adapt to the performance variations over time, co-optimized with scheduling and resource allocation decisions designed to meet real-time deadlines within power constraints to provide maximal QoE. ILLIXR provides a flexible testbed to enable such research.

End-to-End Quality Metrics The end-to-end nature of ILLIXR allows it to provide end-to-end quality metrics such as MTP and see the impact of optimizations on the final displayed images. Our results show that per-component metrics (e.g., head tracking frame rate) are insufficient to determine the impact on the final user experience (e.g., as in Jetson-LP for Sponza). At the same time, there is a continuum of acceptable experiences (as in Sponza for desktop and Jetson-HP). Techniques such as approximate computing take advantage of such applications but often focus on subsystems that make it hard to assess end user impact. ILLIXR provides a unique testbed to develop *cross-layer approximate computing techniques* that can be driven by end-user experience. Our results also motivate work on defining more perceptive quantitative metrics for image quality and ILLIXR provides a testbed for such research.

Other Implications ILLIXR also enables research in designing common compiler intermediate representations (IRs) to enable code generation for a variety of accelerators; device–edge server–cloud server work partitioning; content streaming and multiparty XR; and security and privacy.

Evaluation Tools Architects use a variety of tools to evaluate their research. Design for future domain-specific systems is no different and ILLIXR lays the foundation for other tools as well. ILLIXR can be immediately used for architecture research described above using existing HLS tools to map components/SoC designs to RTL, FPGAs, and ASICs. Given the era of specialization, architecture researchers are increasingly using such methods. We can also use ILLIXR with simulation. We describe three ideas below, and expect that further research into this topic will yield even more solutions. 1) As with other benchmarks, we can scale down ILLIXR inputs and run on simulators such as gem5 [158], with the graphics components on GLTraceSim [159]. 2) We can collect input/output traces of each component via the ILLIXR runtime on a real machine, and organize them like a *rosbag* [160] to drive simulations of components of interest. 3) We can run all ILLIXR components on a common dilated clock – the runtime slows down components running on real hardware to match the real-time speed of simulated components. This models a hybrid real+simulated system. While building such simulators is possible, it is outside the scope of this thesis.

CHAPTER 4: CHARACTERIZATION OF XR COMPONENTS

Chapter 3 motivated specialization of ILLIXR components to meet the performance-power-QoE gap as seen through the end-to-end characterization. This chapter examines the components in isolation to provide insights on how to specialize.

4.1 EXPERIMENTAL METHODOLOGY

We run each component by itself on our desktop and embedded platforms using off-the-shelf standalone datasets: Vicon Room 1 Medium [148] for head tracking, ScanNet `scene0233` [161] for scene reconstruction, OpenEDS [76] for eye tracking, 2560×1440 pixel frames from VR Museum of Fine Art [162] for reprojection and hologram, and 48 KHz audio clips [163, 164] from Freesound [165] for audio encoding and playback. All components other than head tracking and scene reconstruction are input-independent, and thus their behavior does not change with different inputs. We discuss the input-dependence of head tracking and scene reconstruction in the next section.

4.2 HIGH-LEVEL ANALYSIS

Tables 4.1–4.4 summarize the algorithmic tasks within each component, including the key computation and memory patterns, as well as the time spent in each task, analyzed on the high-end desktop platform. We do not show the sensor related ILLIXR components since they are quite simple, and we do not present a table for eye tracking as neural networks are well understood in the architecture community. We define tasks as distinct high level algorithmic steps. Figure 4.1 shows the CPU IPC (Instructions-Per-Cycle) and cycle breakdown for each component in aggregate. We do not provide a similar breakdown on the GPU as GPU profiling tools do not provide one.

Task Dominance No component is composed of just one task. The most homogeneous is audio encoding where ambisonic encoding is 81% of the total – but accelerating just that would limit the overall speedup to 5.3× by Amdahl's law, which may not be sufficient given the power and performance gap shown in Section 3. Head tracking is the most diverse, with seven major tasks and many more sub-tasks.

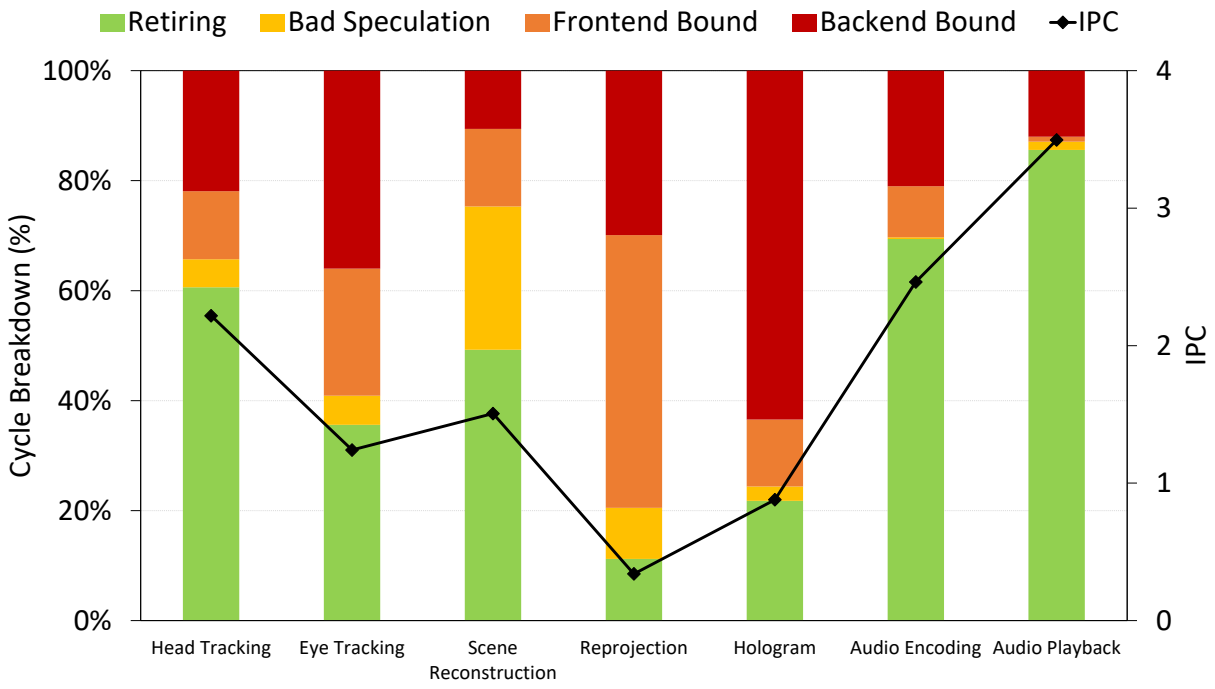


Figure 4.1: Cycle breakdown and IPC of ILLIXR components.

Task Diversity As shown in the component tables, there is a remarkable diversity of algorithmic tasks. These tasks are differently amenable for execution on the CPU (e.g., audio playback), GPU-compute (e.g., hologram), or GPU-graphics (e.g., reprojection). The compute patterns span stencils (KLT), GEMM, Gauss-Newton, and FFT, among others, and are often shared by components (e.g., Cholesky in head tracking and scene reconstruction). The memory patterns are equally diverse, spanning dense and sparse, local and global, and row-major and column-major accesses to various types of data structures. Moreover, the working set sizes of the components range from tens of KB to hundreds of MB, which can change the characteristics of a given memory pattern. Salient examples include locally dense stencils becoming memory bandwidth bound when the working set size is large, and globally sparse accesses *not* becoming memory bound when the working set size is small relative to the on-chip storage capacity.

Microarchitectural Diversity Figure 4.1 shows significant microarchitectural diversity in addition to the algorithmic diversity discussed above. The IPC ranges from 0.3 (reprojection) to 3.5 (audio playback), with a variety of bottlenecks on the frontend and backend of the CPU pipeline.

Table 4.1: Task breakdown of head tracking.

Task	Time	Computation	Memory Pattern
Feature detection Detects new features in the new camera images	15%	KLT; FAST	Globally mixed dense and sparse image accesses; locally dense image stencil
Feature matching Matches features across images	13%	KLT; GEMM; linear algebra	Globally mixed dense and sparse image accesses; locally dense image stencil; mixed dense and random feature map accesses
Feature initialization Adds new features to state	14%	SVD; Gauss-Newton; Jacobian; nullspace projection; GEMM	Dense feature map accesses; mixed dense and sparse state matrix accesses
MSCKF update Updates state using MSCKF algorithm	23%	SVD; Gauss-Newton; Cholesky; QR; Jacobian; nullspace projection; chi^2 check; GEMM	Dense feature map accesses; mixed dense and sparse state matrix accesses
SLAM update Updates state using EKF-SLAM algorithm	20%	Identical to **MSCKF update**	Similar, but not identical, to **MSCKF update**
Marginalization Removes features from state	5%	Cholesky; matrix arithmetic	Dense feature map and state matrix accesses
Other Miscellaneous tasks	10%	Gaussian filter; histogram	Globally dense image stencil

Input-Dependence Although we saw significant per-frame execution time variability in all components in the integrated system, the only components that exhibit such variability standalone are head tracking and scene reconstruction. Head tracking shows a range of execution time throughout its execution, with a coefficient of variation from 17% to 26% across Vicon Room 1 Easy, Medium, and Hard [148]. Across several ScanNet sequences, scene reconstruction's execution observes even higher variation, depending upon the depth complexity of the scene and the movement of the camera.

Table 4.2: Task breakdown of scene reconstruction.

Task	Time	Computation	Memory Pattern
Image Processing Pre-processes RGB-D image for tracking and mapping	10%	Bilateral filter; Box filter; Gaussian filter	Globally dense image stencil
Pose Estimation Estimates 6DOF pose	80%	Iterative closest point; Cholesky; reduction	Globally mixed dense and sparse image accesses; locally dense image accesses
Visibility Check Calculates visible voxels in current frame	2%	Raymarching; matrix transformations; parallel prefix sum	Globally dense accesses to depth image and visibility list; random accesses to voxel map
Voxel Fusion Updates visible voxels with new depth information	2%	Frame projective TSDF calculation (Eq. 6 [81]) TSDF and weight update (Eq. 11 and 12 [81])	Random accesses to voxel map; locally dense accesses to voxel blocks
Raycasting Generates reconstructed image	6%	Raymarching; software rasterization; trilinear interpolation; vertex and normal map generation	Random accesses to voxel map; locally dense accesses to voxel blocks; globally dense accesses to reconstructed image

4.3 ARCHITECTURAL DEEP DIVE

We also perform a detailed microarchitectural characterization of each component task. Below we highlight salient characteristics of each component.

Head tracking is a complex CPU workload with wide microarchitectural task diversity. The average IPC is 2.2, but there are several computationally intensive tasks with good vectorization that present higher average IPC; e.g., KLT and GEMM have an IPC of 3.2+. On the memory side, the working set size of most tasks is several hundred KBs, which is larger than the L1 and L2, but is small enough to fit in the LLC – the LLC shows only 0.1 MPKI (misses per kilo-instruction) while the L2 shows 7.9 MPKI. Our results include the effect of demand prefetchers in the CPU, which are quite effective – although the L2 shows 7.9 MPKI overall, it shows 0.6 MPKI for demand loads.

Eye tracking is a typical deep neural network that spends 74% of its total time executing convolution operations and activation functions, 19% performing batch copies, and the

Table 4.3: Task breakdown of visual pipeline components.

Task	Time	Computation	Memory Pattern
		Reprojection	
FBO FBO state management	24%	Framebuffer bind and clear	Driver calls; CPU-GPU synchronization
OpenGL State Update Sets up OpenGL state	54%	OpenGL state updates; one drawcall per eye	Driver calls; CPU-GPU synchronization
Reprojection Applies reprojection transformation to image	22%	6 matrix-vector MULs/vertex	Dense accesses to uniform, vertex, and fragment buffers; 3 sparse texture accesses/fragment
		Hologram	
Hologram-to-depth Propagates pixel phase to depth plane	57%	Transcendentals; FMADDs; tree reduction	Globally dense accesses to hologram phases
Sum Sums phase differences from hologram-to-depth	$<$ 0.1%	Tree reduction	Globally dense accesses to partial sums
Depth-to-hologram Propagates depth plane phase to pixel	43%	Transcendentals; FMADDs; thread-local reduction	Globally dense accesses to depth phases

remaining 7% in miscellaneous tasks. Notably, the weights only occupy 0.98 MB but a total of 1922 MB is accessed during a forward pass, still making it memory bandwidth bound. However, the batch size is only two (one image per eye), resulting in low overall utilization of the GPU.

Scene reconstruction is a hybrid CPU-GPU workload that spends most of its time on GPU tasks and specifically in pose estimation. It is a memory bandwidth bound workload with many tasks consuming 200 GB/s of memory bandwidth and a few even surpassing 400 GB/s due to the large number of accesses per pixel (depth, normal, vertex, etc.). Reductions are a commonly found compute primitive in many sub-tasks (ICP and all of the various filters). Inter-frame reuse is significant, as 80%–90% of the voxel map is reused from frame to frame. Intra-frame reuse, however, has significant variation as some kernels possess no reuse because of streaming accesses (such as voxel fusion) while others possess excellent reuse

Table 4.4: Task breakdown of audio pipeline components.

Task	Time	Computation	Memory Pattern
		Audio Encoding	
Normalization INT16 to FP32	7%	Element-wise FP32 division	Globally dense accesses to audio samples
Encoding Sample to soundfield mapping	81%	$Y[j][i] = D \times X[j]$	Globally dense column-major accesses to soundfield
Summation HOA soundfield summation	12%	$Y[i][j] += X_k[i][j]\ \forall k$	Globally dense row-major accesses to soundfield
		Audio Playback	
Psychoacoustic filter Applies optimization filter	29%	FFT; frequency domain convolution; IFFT	Butterfly pattern; Globally dense accesses to FFT output
Rotation Rotates soundfield using pose	6%	Transcendentals; FMADDs	Globally dense accesses to soundfield
Zoom Zooms soundfield using pose	5%	FMADDs	Globally dense column-major accesses to soundfield
Binauralization Applies HRTFs	60%	Identical to **psychoacoustic filter**	Identical to **psychoacoustic filter**

due to local stencil operations (such as bilateral filtering). The depth image has the highest reuse as it is accessed in four out of the five tasks. While accesses to the voxel map are irregular, they are not a bottleneck as they constitute an extremely small fraction of the entire component. This is because they only occur along the critical path in visibility check, a task which consumes only 2% of total frame time.

Reprojection is a hybrid CPU-GPU workload that spends a significant amount of time setting up framebuffer and OpenGL state prior to execution of the reprojection shaders. Framebuffer and OpenGL state updates are both performed via the GPU driver. Consequently, reprojection only has an IPC of 0.3, with most of the CPU cycles spent in frontend stalls due to the large instruction footprint of the GPU driver. The shaders themselves have very little compute and are memory bandwidth bound due to the size of the framebuffer. Moreover, the irregular texture cache accesses due to chromatic aberration correction result

in a cache hit rate of 75%, which is low compared to the 90% achieved by non-correcting shaders.

Hologram executes all its tasks on the GPU as CUDA kernels, all of which are compute-bound. Almost all the executed instructions are FP fused multiply-add (FFMA), integer multiply-add (IMAD), and FP add (FADD). There is significant utilization of the FP64 hardware pipe – 75% in hologram-to-depth – as the transcendental calculations necessitate high precision. Memory traffic is limited to accessing each pixel's phase value only once in each task (reads in hologram-to-depth and writes in depth-to-hologram); however, this is significant data given the size of the framebuffer, and consumes 75-140 GB/s of memory bandwidth. The depth plane's phase values are accessed more often, but are only few tens of bytes in size, and are stored in the scratchpad for reuse.

Audio encoding is a compute-bound workload that is able to leverage vectorization and dense data structure accesses to achieve an IPC of 2.5, with 69% of cycles being spent retiring instructions. However, the component's IPC is limited due to backend stalls caused by division and modulo operations being bottlenecked by the lone hardware divider.

Audio playback is also a compute-bound workload, but has no division operations, resulting in even higher IPC and hardware utilization. The FFT and FMADD compute is vectorized, and the densely accessed 64 KB HOA soundfield comfortably fits in the L2 cache of the CPU. Consequently, 86% of the cycles are spent retiring instructions, and a fairly high IPC of 3.5 is achieved.

4.4 KEY IMPLICATIONS FOR ARCHITECTS

The diversity of components and tasks, the lack of a single dominant task for most components, and per-frame variability pose several challenges for hardware specialization. It is likely impractical to build a unique accelerator for every task given the large number of tasks and the severe power and area constraints for XR devices: leakage power will be additive across accelerators, and interfaces between these accelerators and other peripheral logic will further add to this power and area (we identified 27 tasks in across all components and expect more tasks with new components).

At the same time, our results show that a number of common primitives exist across components; e.g., Cholesky in head tracking and scene reconstruction, making the case for *shared hardware across components.* While determining whether components share compute or memory primitives is possible by analyzing the components in isolation, whether we should instantiate only one Cholesky block or whether we should duplicate or pipeline it to meet the requirements of both components can only be answered by taking run-time interactions

into account, which ILLIXR enables.

The choice of shared hardware not only impacts architecture (and the software stack such as compilation and scheduling), but also microarchitecture. Analyzing head tracking in isolation, the communication between GEMM and Cholesky as part of the chi^2 check can be hardcoded to improve efficiency. In a full system, the shared nature of Cholesky motivates a different microarchitecture with support for efficient communication with multiple producers and consumers instead of just GEMM. The above observations motivate new *design space exploration tools research for system-wide codesign.*

Our results also motivate *automated techniques for determining what to accelerate.* An open question is what is the granularity of acceleration; e.g., one accelerator per task or a more general accelerator that supports *common primitives shared among components.* Besides the design of the accelerators themselves, architects must also develop *efficient on-chip memory hierarchies* that are able to accommodate a wide range of memory access characteristics, and determine *accelerator communication interfaces* to implement efficient communication.

Finally, although not explicitly studied here, algorithms for the various components continue to evolve and there are multiple choices available at any time (ILLIXR already supports such choices). This motivates *programmable accelerators* with the attendant challenges of developing a software stack for such a system. ILLIXR provides a testbed to perform all of the above research in the context of the full system.

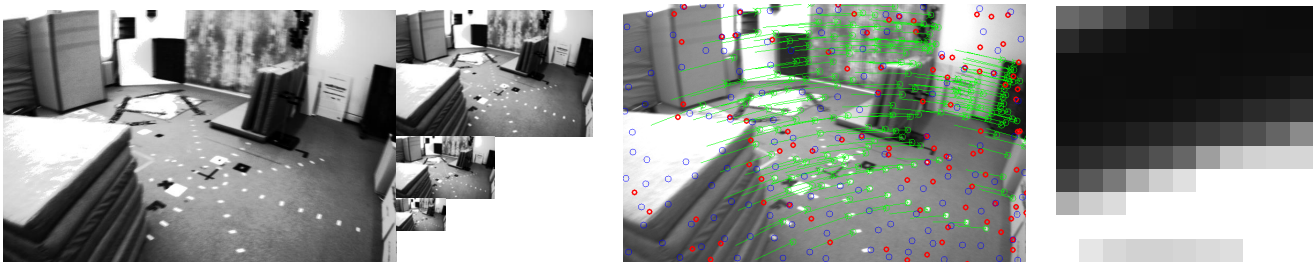
CHAPTER 5: DISTRIBUTED ON-SENSOR COMPUTING

Modern XR devices contain a large number of camera sensors to aid head, hand, eye, and face tracking, scene reconstruction, and other features. For example, the Oculus Quest Pro has ten cameras on the headset [166], and several more on the hand controllers for various tracking tasks. Head tracking alone requires four or more cameras in most headsets [11]. As a result, the power consumption of I/O between the cameras and the SoC has become non-trivial, especially since cameras connect to the SoC via MIPI CSI-2 [167], a link that consumes tens of pJ per bit to transmit data.

Indeed, the system-level power analysis of ILLIXR in Chapter 3 showed I/O power to be a significant contributor to total system power. Sys power, which includes display, storage, and I/O, consumes up to 30% of total power on Jetson-LP (Section 3.3). A part of I/O power comes from the communication between the tracking cameras and head tracking, as the cameras ship full images to the SoC in each frame. A design principle for reducing I/O power is on-sensor computing, in which a computation or its subset is performed on the camera sensor itself. Doing so reduces the amount of data that needs to be transmitted over the power hungry I/O link.

In this chapter, we develop a distributed on-sensor architecture for VIO to showcase the applicability of on-sensor computing to XR. We assume that VIO has already been accelerated in hardware [168, 169, 170] due to its high CPU utilization, as shown in Chapter 3, and its importance in driving the entire system, as discussed in Chapter 2. Our design replicates the point tracking frontend of VIO within each camera sensor and provides a simple centralized directory to coordinate the execution among the different cameras. As a result, full images never leave the cameras. This reduces power consumption due to I/O by 37×, VIO chip power by 3.3×, and total VIO subsystem power by 1.3×. This chapter also highlights the need to look at sensors when designing architectures for future XR devices, as sensors constitute 87% of the remaining power.

The solutions discussed in this chapter can be extended to other components of an XR system which possess similar image processing characteristics as VIO, such as eye tracking [14] and hand tracking [15]. Although not explored by us, our design has implications for security and privacy and system design as well. For instance, raw pixels are consumed within the camera itself and not transmitted over the I/O link, potentially reducing the attack surface. Furthermore, due to reduced off-sensor traffic, system designers may have more flexibility in terms of communication link choice (e.g., I3C [171]) and wire placement (e.g., less congestion due to lack of central chip).



(a) Image Pyramid (b) Point Tracking (c) Point Patch

Figure 5.1: (a) Image pyramid with four levels. Each level has half the resolution in each dimension compared to the previous level. (b) Tracked Shi-Tomasi points in the base pyramid level. Successfully tracked points are shown in green, with a green line denoting how the point was tracked over time. Newly detected points are shown in blue, and points which were detected in previous frames but could not be tracked successfully in this frame are shown in red. (c) 11×11 pixel patch of a successfully tracked point.

5.1 BACKGROUND

A VIO algorithm can be divided into a point tracking frontend and a pose estimation backend. The frontend is responsible for finding salient features in camera images and tracking them from one image to the next. The backend is responsible for using IMU readings and the feature information provided by the frontend to estimate the user's pose.

5.1.1 Point Tracking Frontend

The first step in the tracking process is typically the generation of an image pyramid, where representations of the image at progressively smaller scales are created. For instance, a 640x480 image may have an image pyramid with images of resolution 640x480 at level 0, 320x240 at level 1, 160x120 at level 2, and so on. A common way of generating an image pyramid is to apply Gaussian blur to a level and then downsample it [172]. An example image pyramid is shown in Figure 5.1a. Image pyramids are used during tracking (described below).

Once the image pyramid has been generated, *features* are detected in the image. Features are salient points that correspond to edges or corners in the image. Edges are simply pixels where there are discontinuities in brightness, contrast, depth, etc., and corners are intersections of two edges. Edges and corners capture real world features that are easy to track from one frame to another, such as the edge of a desk or a highly textured sofa (as opposed to a textureless wall). FAST, Harris, Shi-Tomasi, etc. are all examples of commonly used feature detectors [173]. Figure 5.1b shows various Shi-Tomasi corners.

Once features have been detected, their *descriptors* are calculated, which capture basic characteristics of the feature, such as its shape, size, color, etc. in a binary format. This aids in feature matching, where the descriptors of two features can be compared to determine whether they are the same feature or not. Commonly used feature descriptors are BRIEF, SIFT, ORB, etc., and differ in terms of properties such as robustness, scale invariance, and rotation invariance [173]. In addition to descriptors, based on the tracking approach (see below), the *patch* of a feature is also extracted. A patch is a small block of pixels surrounding the detected feature. An example patch is shown in Figure 5.1c for one of the tracked points in Figure 5.1b.

The tracking of features from frame to frame is a correspondence search problem, which can be tackled in a variety of ways. Optical flow and descriptor matching are two commonly used approaches [69, 174]. The former relies on matching image patches and the latter relies on matching feature descriptors. Matching patches across multiple pyramid levels improves tracking accuracy. In this chapter, we assume descriptor matching for initialization and patch matching otherwise.

After all of the above steps have been performed, the frontend provides a set of points to the backend. We define a *point* as consisting of the following information: latest depth estimate, descriptor, patch, the score or "goodness" of the point, and a list of *observations* – the ID of the camera(s) the point was observed from, the level of the pyramid it was detected in, a camera ray towards the point, and an estimate of the distance along the ray. The output of the frontend includes points from previous frames that were successfully matched in the new frame, points that were not successfully matched, and new points that were detected in the new frame. The backend may choose to use only some of this information.

5.1.2 Pose Estimation Backend

The pose estimation backend takes the information from the point tracking frontend and the IMU, and estimates the 6 Degrees-of-Freedom (6DOF) *pose* of the user – 3 degrees for the location and 3 degrees for the orientation. The estimation is performed using either a filtering-based approach or an optimization-based approach [69, 174]. Both approaches use observations of the points and integrated acceleration and velocity readings from the IMU to estimate the pose. The backend also includes a point selection mechanism that provides feedback about which points to track to the frontend. This feedback can include whether to discard any points if their score is low, how many new points to detect, and so on. The backend also sends camera calibration parameters to the frontend (if it performs online camera calibration). The ideas described in this chapter are agnostic to the choice of

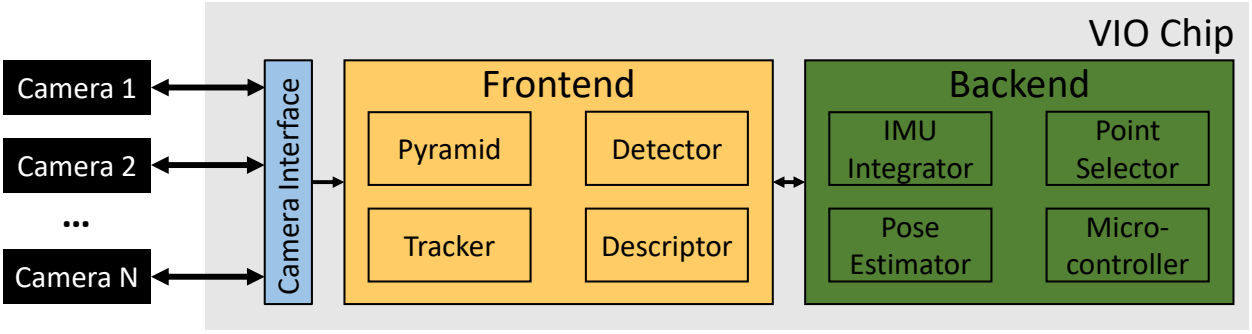


Figure 5.2: Centralized VIO architecture with frontend and backend marked in yellow and green, respectively.

pose estimation backend.

5.2 DESIGN

We first describe a general hardware architecture for a traditional centralized design. We then propose a simple distributed design, and then describe progressive optimizations to arrive at the final distributed VIO architecture. Although we do not explore the security, privacy, and system design implications of our design in this chapter, we nonetheless highlight the design decisions that may affect security, privacy, and system design.

5.2.1 Centralized Baseline

We assume the baseline configuration to be a specialized fully hardened implementation of VIO where multiple cameras connect to a centralized chip via MIPI CSI-2 [167], as shown in Figure 5.2. Each camera sends raw pixels to this chip, which are then used by the point tracking frontend to track existing points and find new ones. The pose estimation backend uses the observations from the tracker and readings from the IMU to estimate the 6DOF pose and for calculating which points to track in the next frame. The microcontroller is used for configuring the system and for housekeeping tasks (e.g., debugging).

This design suffers from both high link power due to image transfers between the cameras and the VIO chip, as well as potentially weak privacy as raw images are transmitted over a communication link that a programmable CPU could theoretically access. Furthermore, the chip is a centralized point of failure.

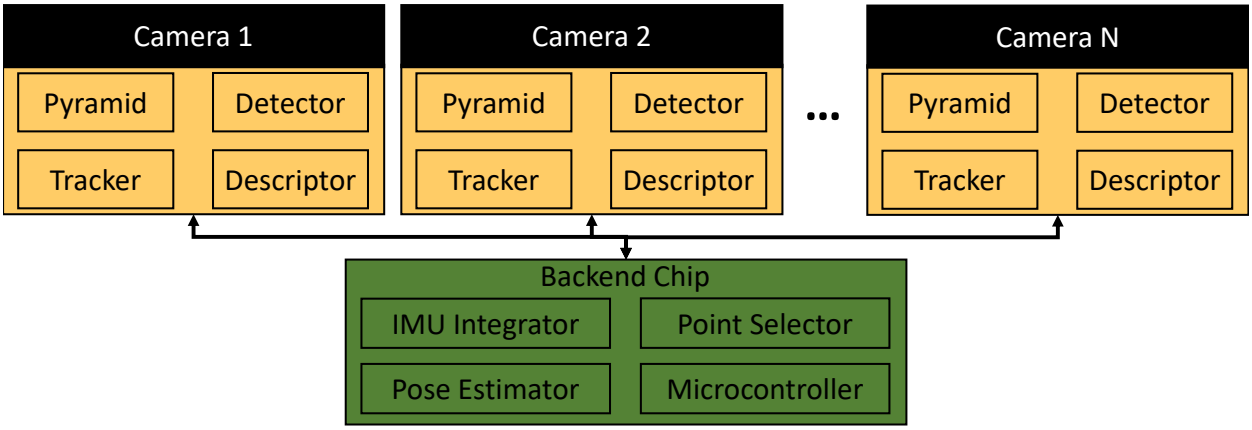


Figure 5.3: Distributed VIO architecture with on-sensor point tracking. The frontend chips are in yellow while the backend chip is shown in green.

5.2.2 On-Sensor Point Tracking

Our main insight from analyzing the centralized design is that it is only the point tracking frontend that requires access to entire camera images – once features have been detected and their descriptors and patches have been obtained, the full image is no longer required, either by the frontend or the backend (which only operates on features and their observations). Based on this insight, we propose a distributed on-sensor VIO architecture in which the entire point tracking frontend is moved into each camera, as shown in Figure 5.3. We refer to the on-sensor tracking frontend as the *frontend chip*. We assume a technology similar to Sony [175], where compute logic can be integrated into the camera chip itself using 3D stacking, thereby providing the point tracking frontend access to raw pixel data without leaving the frontend chip.

The remaining parts of VIO, primarily the pose estimator and its associated blocks, which we call the *backend chip*, remain in the centralized chip. Importantly, the point memory (which stores patches, descriptors, etc.) remains in the backend. Each frame, the backend chip requests the frontend chip to track a set of points. For each point, it sends the entire point structure to the frontend. When the frontend finishes tracking old points and detecting new ones, it sends back the set of points to the backend. In addition to points, the backend also sends pose and camera calibration data to the frontend. Since the frontend and the backend only exchange points and not entire images in this design, data movement is significantly reduced. Moreover, privacy could be improved as well.

5.2.3 Communication Pruning

We further optimize data movement by noting that the frontend chips and the backend chip do not need to exchange the entire point data structure when communicating. For instance, the backend does not need to send the point descriptors when notifying the frontend of which points to track (but still needs to send the patches). Similarly, the frontend does not need to send some of the point metadata to the backend as it is only used internally by the point tracker. In this iteration of our design, we further minimize data movement between the frontend and the backend by pruning this unnecessary data communication. After pruning, the communication that occurs is as follows.

Backend to Camera (B2C): For each point that the backend requests an on-sensor frontend to track, it sends the frontend the point's ID, depth estimate, and patch. The backend also sends the latest pose and camera calibration information once per frame.

Camera to Backend (C2B): For each of the incoming points to be tracked, the on-sensor frontend sends back observations (defined in Section 5.1.1), and for each of the newly generated points, the entire point is sent back to the backend.

5.2.4 Caching

Overview The backend only needs to know which points have to be tracked and what their observations are. It does not need to know anything about patches at all. Based on this insight, the next configuration adds a patch cache to each frontend. Whenever a frontend detects a new point, the point's patch is inserted into the cache, which is indexed using the point's ID. In this design, the backend and the frontends only communicate via these IDs, which are only a few bytes in size.

When the backend notifies a frontend to track a particular point, the frontend looks up the point's patch in its cache. On a cache hit, the patches are read from the cache and a significant amount of data movement is saved as no patches are transferred. On a cache miss, the frontend notifies the backend of the miss. The backend then looks up in a directory which frontend owns the patch in question, and requests that frontend to send the patch to the desired frontend. As patches are read-only, they can be in multiple caches at the same time.

Note that the per-frontend caches only store patches, and other point data (descriptors, etc.) is still stored in the backend. Observations and descriptors of new points still have to be sent from each frontend to the backend, and pose and calibration data from the backend to the frontends. The final design is shown in Figure 5.4.

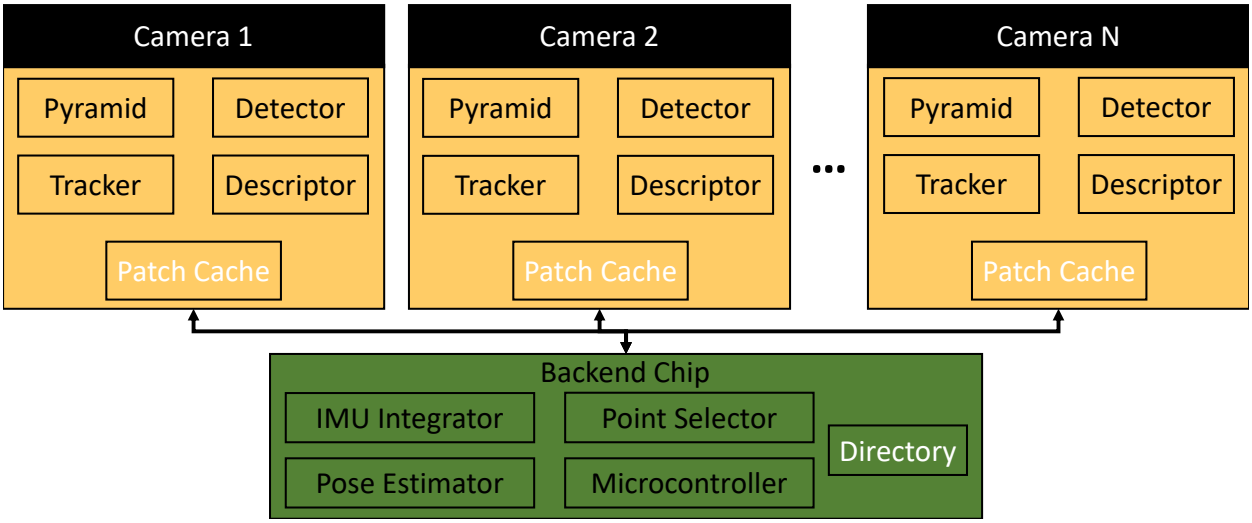


Figure 5.4: Distributed VIO architecture with patch caching. The new components, the patch cache and the directory, are shown in white.

Details To cache patches, we add a simple direct-mapped cache to each frontend, as shown in Figure 5.4. This cache is indexed using unique point IDs and each entry consists of the point's patch.

The caches are managed by a central directory in the backend. The directory keeps track of which patches reside in which cache – called a sharers list – and which sets (entries) are occupied in each cache — called an occupancy table. Since the backend does not ever need patches and only needs to know where they are, there is no centralized patch memory in the backend anymore. Instead, the combined capacity of all the caches is used to track all the points — a maximum of 512 points in our architecture.[9]

Whenever new points are created by a frontend, their patches are inserted into the local cache and the directory is notified. When the frontend needs a patch and it hits in the cache, nothing needs to be done. On misses, however, the directory is notified, which then uses the sharers list to determine which cache/camera has the patch and asks it to send the patch to the original requester.

As mentioned above, the combined capacity of all the caches must be sufficient to track all the points. That is, if we wish to track a total of 512 points (and therefore 512 patches) and we have four caches, each cache must contain at least 128 patches. If there are exactly 128 entries per cache, no duplicates are allowed, and patches can at most be in one cache at any given time. Consequently, on a patch miss, the current owner has to remove the patch from its cache before sending it to the requester. Similarly, conflicting entries in a cache have to

[9]We chose 512 points as a conservative upper bound based on analysis of our internal tracking datasets.

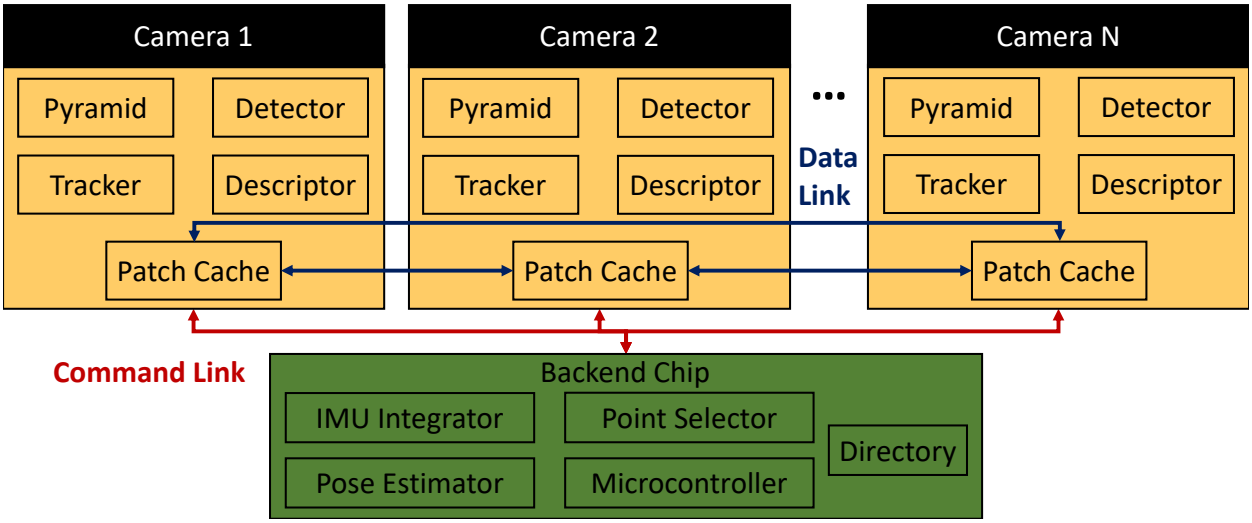


Figure 5.5: Distributed VIO architecture with separate data and command links. The command link is shown in red while the data link is shown in dark blue.

be relocated elsewhere in order to ensure there are no duplicates. The directory performs this task by using the occupancy table and relocates conflicting entries into empty slots in remote caches.

Not allowing duplicates has a detrimental effect on hit rate — many points are shared between cameras and this causes them to ping-pong between caches. By increasing the capacity of the caches, duplicates can be allowed. However, increasing cache capacity also increases leakage power and thus a trade-off must be made between energy saved from data movement and energy added from leakage. We explore this trade-off in Section 5.4.2.

Hit Rate Optimization A possible optimization for improving cache hit rate without increasing cache capacity (and thus leakage power) is to allow duplicates as long as there is sufficient capacity remaining in the caches. The intuition behind this idea is that even though the maximum tracking capacity of the system may be 512 points, far fewer points may actually be getting tracked in any given frame, resulting in wasted storage. If that storage can be used for duplicates, ping-ponging of patches can be reduced, increasing cache hit rate and decreasing communication power. However, we chose not to implement this optimization as it appeared to provide only marginal gains, as shown in Section 5.4.3.

5.2.5 Split Communication Links

Even though the design presented in Section 5.2.4 prevents raw pixels from leaving the cameras, it still passes around patches, which the programmable embedded CPU can readily

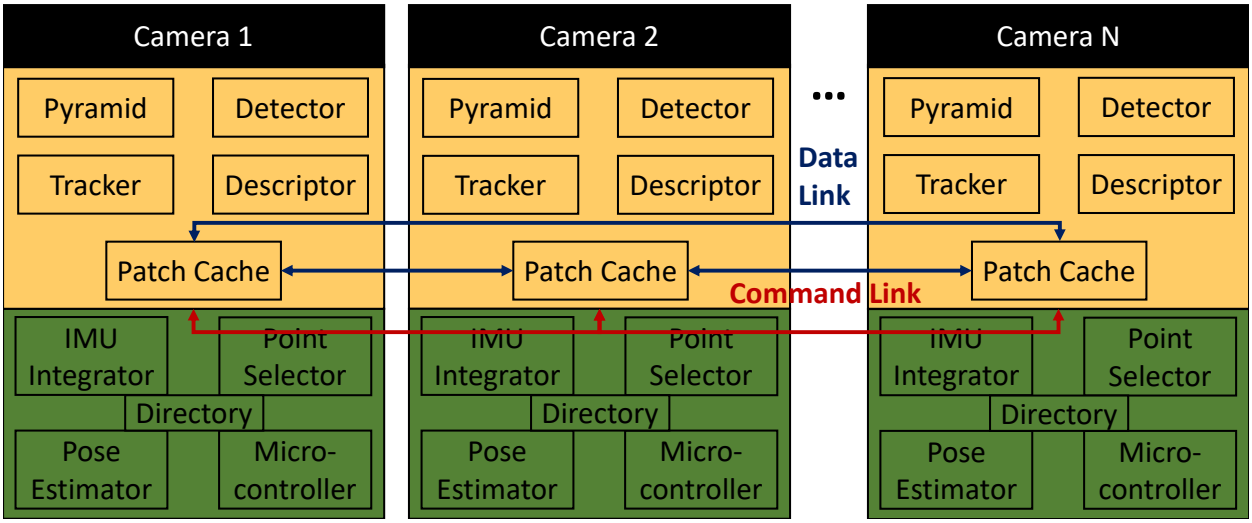


Figure 5.6: Distributed VIO architecture with fault tolerance. Each camera sensor houses an identical chip.

access. If the CPU gets hacked, a collection of patches can provide a fairly representative reconstruction of the user's surroundings [176]. To tackle this problem, we split the communication link into two separate links: command and data.

The command link is shared across all the chips and is only used for exchanging metadata and requests/responses. The data link is used for transferring patches between the frontends, and the backend does not have access to it. That is, the backend physically does not have wires going to that link, and thus cannot directly access the patches even if a malicious party were to obtain access to the CPU. The separate links are shown in Figure 5.5.

5.2.6 Chip Replication

In a centralized design, chip failure could mean complete loss of pose tracking. Even in our distributed design, the backend is a central point of failure. To alleviate this concern, we replicate the backend in each camera and shut off all but one of the backends. If the active backend fails, another can take over (this would require VIO re-initialization, which may result in a temporary loss of tracking), until no chips are left. An important benefit of this approach is that only one homogeneous chip needs to be designed, and not a separate one for the frontend and a separate one for the backend. This can save significant engineering effort, which is often a primary concern in practice. Our final distributed on-sensor VIO architecture is shown in Figure 5.6.

5.3 EXPERIMENTAL METHODOLOGY

We evaluate the first four configurations described in Section 5.2: centralized, distributed, distributed with pruning, and distributed with both pruning and caching . We do not evaluate the two configurations with split communication links and chip replication since those two configurations only affect privacy, security, and engineering cost, and have no bearing on link power, the metric which we are concerned with in this chapter.

We instrument a custom VIO implementation to collect communication data. For the cache-based configuration, we implement the cache at a functional level inside the VIO code, and used a 256-entry cache in all our experiments unless otherwise stated. All reported results are over ∼300 hours of video taken from four tracking cameras, spanning a variety of use cases. Since the computation remains unchanged, we observe no difference in pose estimation accuracy metrics, such as Average Trajectory Error and Relative Pose Error [121], between the centralized and distributed designs.

We obtain time and dynamic and leakage power estimates by using RTL implementations of the various computational blocks of our algorithm. To compute link power, we assume a 50 pJ/bit MIPI CSI link between the cameras and the VIO chip, in line with prior work [177, 178]. We do not report timing information as both the centralized and distributed designs operate several factors faster than the target camera frame rate. Moreover, we do not expect a performance loss in the distributed designs as the amount of work performed remains unchanged, and transferring tens of KBs due to cache misses over a 1 Gbps MIPI link would consume less than 0.5 ms. Finally, we do not include directory power as it is negligible (the directory is only a few Kilobits in size).

We consider power at three different levels: link, VIO chip, and VIO subsystem. Link power is the power of the MIPI CSI link. VIO chip power includes the VIO accelerators, compute and memory leakage, and link power. VIO subsystem power contains camera and IMU power in addition to VIO chip power.

5.4 RESULTS

5.4.1 Traffic Reduction

Figure 5.7 shows the link traffic breakdown of the four designs listed in Section 5.3 normalized to the baseline centralized design. Since the distributed designs consume only a small percentage of the baseline link traffic, the y-axis of the graph is capped at 16% to clearly show the traffic breakdown. As can be seen, even the basic distributed partitioning reduces

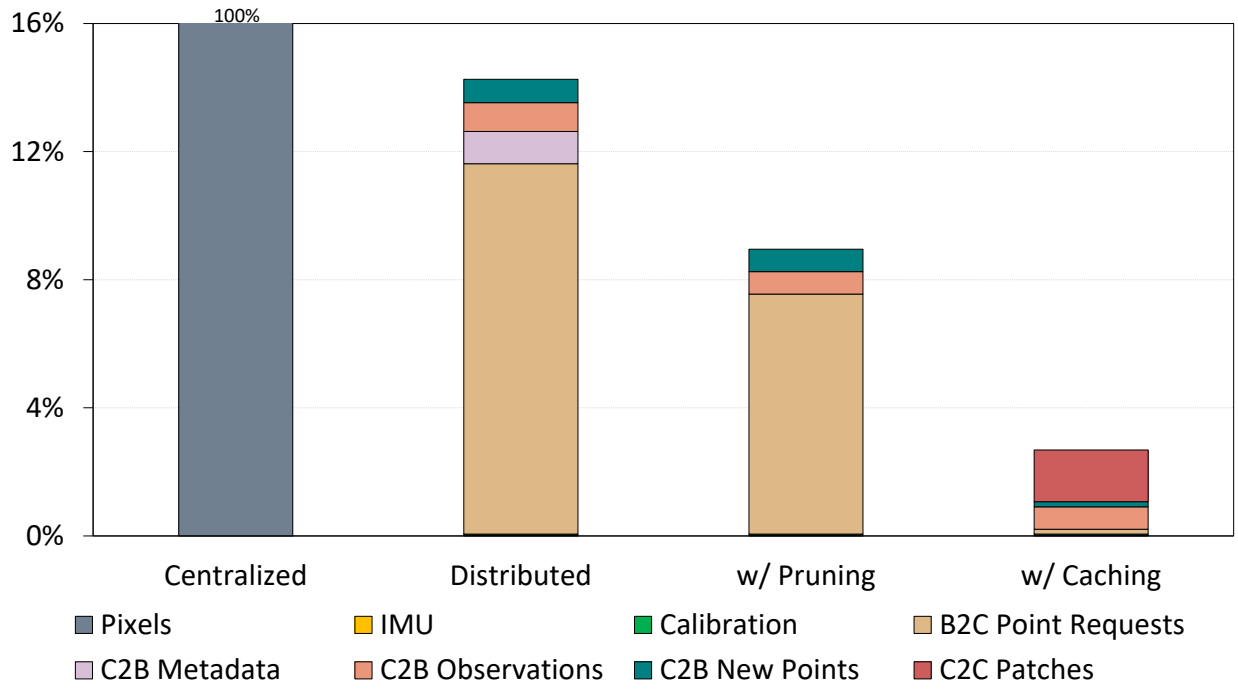


Figure 5.7: Normalized link traffic of each VIO architecture. The y-axis is capped at 16% for clarity. B2C denotes backend-to-camera communication, C2B denotes camera-to-backend communication, and C2C denotes camera-to-camera communication.

communication traffic per frame by 7×. Pruning provides an additional factor of 1.6× improvement and caching another factor of 3.3× on top of that, for a total reduction of 37×. The non-cache-based distributed designs are dominated by backend-to-camera point request traffic, which contains the patches of the points that need to be tracked. The cache-based design no longer requires such communication.

Instead, the cache-based design's traffic is dominated by cache-to-cache patch transfers (60%). While this may seem to motivate future work into improving the hit rate of our caches, link power is already sufficiently low (Section 5.4.3) that any extra effort would likely provide diminishing returns.

5.4.2 Cache Size Scaling

As mentioned in Section 5.2.4, there exists a trade-off between power saved due to less data movement and power added due to leakage when increasing the size of the patch cache. To study this trade-off, we performed a cache size scaling study. Figure 5.8 shows link, leakage, and total power on the left y-axis, and cache hit rate on the right y-axis for four different cache sizes. The cache sizes are shown in terms of the number of points that they

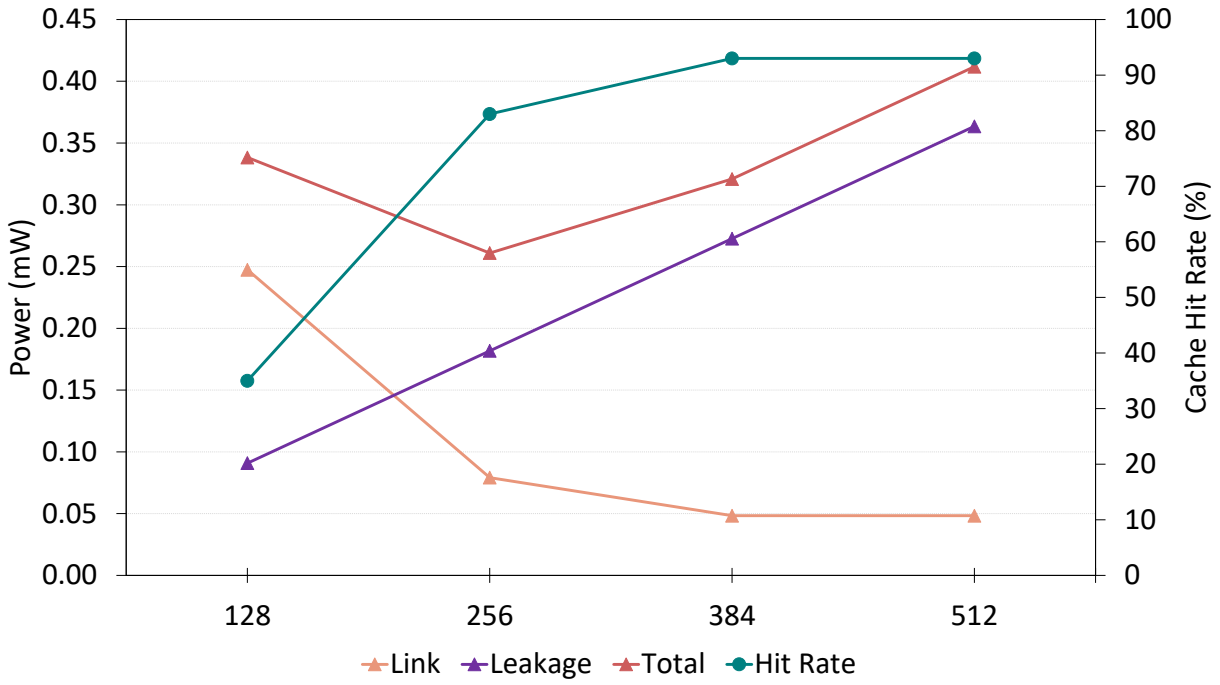


Figure 5.8: Link, leakage, and total power (triangles, left y-axis), and cache hit rate (circles, right y-axis) for various cache sizes. The 256-entry cache achieves the best balance between link and leakage power.

can cache. The 128-entry cache disallows duplicates, whereas the 256-, 384-, and 512-entry caches allows two, three, and four duplicates, respectively, of each patch across the entire system.

As shown in Figure 5.8, larger caches have higher hit rates and therefore lower link power. The 384-entry cache has a hit rate of 93%, with the majority of the remaining misses coming from newly allocated points in other cameras. The hit rate does not improve with a 512-entry cache because four duplicates are never required in practice. At the same time, larger caches also have higher leakage power, and the added leakage eventually offsets the benefits of a high hit rate. Ultimately, the 256-entry cache achieves the lowest overall power (sum of link power and cache leakage power), despite its modest hit rate of 83%.

5.4.3 Power Reduction

Figure 5.9 shows the power breakdown of the VIO chip in terms of accelerator power (VIO), compute and leakage power, and link power for the four different configurations. As can be seen, link power constitutes 74% of VIO chip power in the baseline centralized design. The link traffic reductions achieved by the three distributed designs significantly

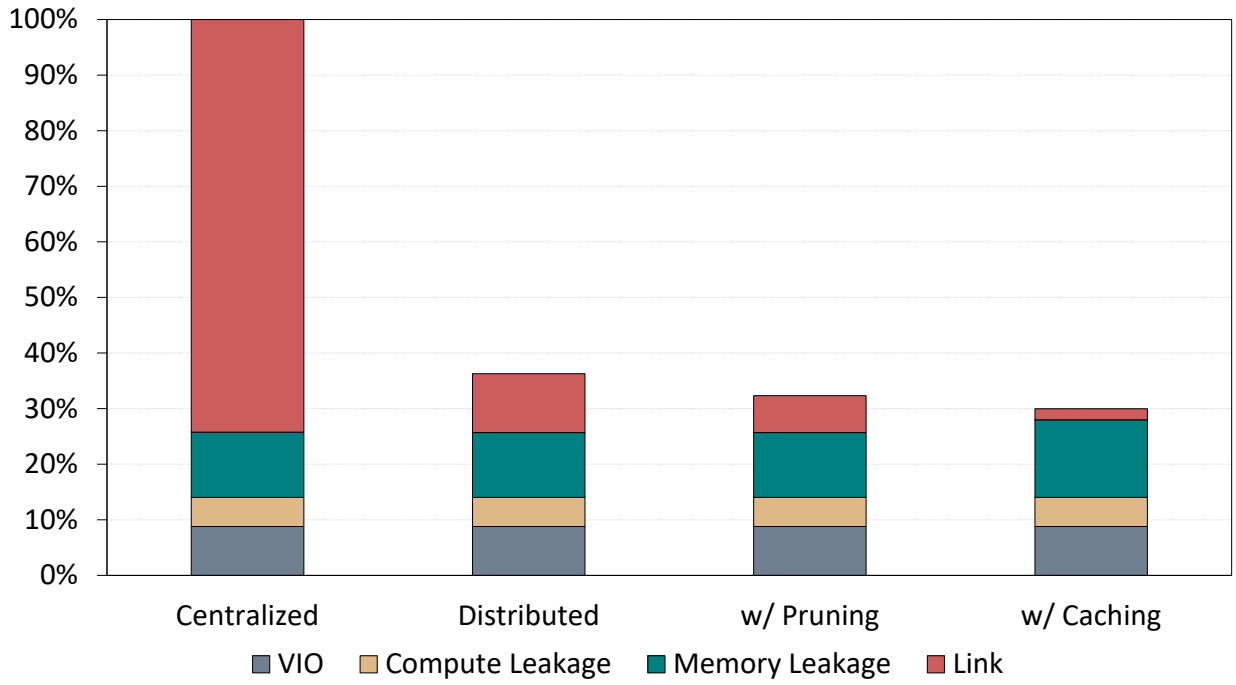


Figure 5.9: Normalized power breakdown of the VIO chip for the four VIO designs. All bars are normalized to the VIO chip power of the baseline centralized design.

reduce this fraction. The final cache-based design reduces link power to just 7% of the remaining total, resulting in a 3.3× VIO chip power reduction compared to the centralized design. While the power reduction benefits of caching may seem minimal compared to a non-caching solution, as discussed in Section 5.2.5, a cache-based design may afford privacy and security that a non-cache-based design may not. The remaining power is dominated by compute and memory leakage. The cache-based design has higher memory leakage than the non-cached-based design, but offsets it by reducing link power more significantly. This motivates devising a better cache architecture in the future which aims to achieve similar hit rates with a smaller capacity, reducing memory leakage power.

Figure 5.10 shows the normalized power breakdown of the entire VIO subsystem in terms of VIO chip components, the IMUs, and the cameras for the four different VIO designs. When considering the entire VIO subsystem, link power constitutes 25% of total power, as opposed to 74% when considering just the VIO chip. By Amdahl's law, completing removing link power would result in a 1.33× reduction in total power. Our caching based design almost achieves this ideal target, with a power reduction of 1.31×. Of the remaining power, 87% is consumed by the cameras and IMUs, motivating research in more efficient sensors.

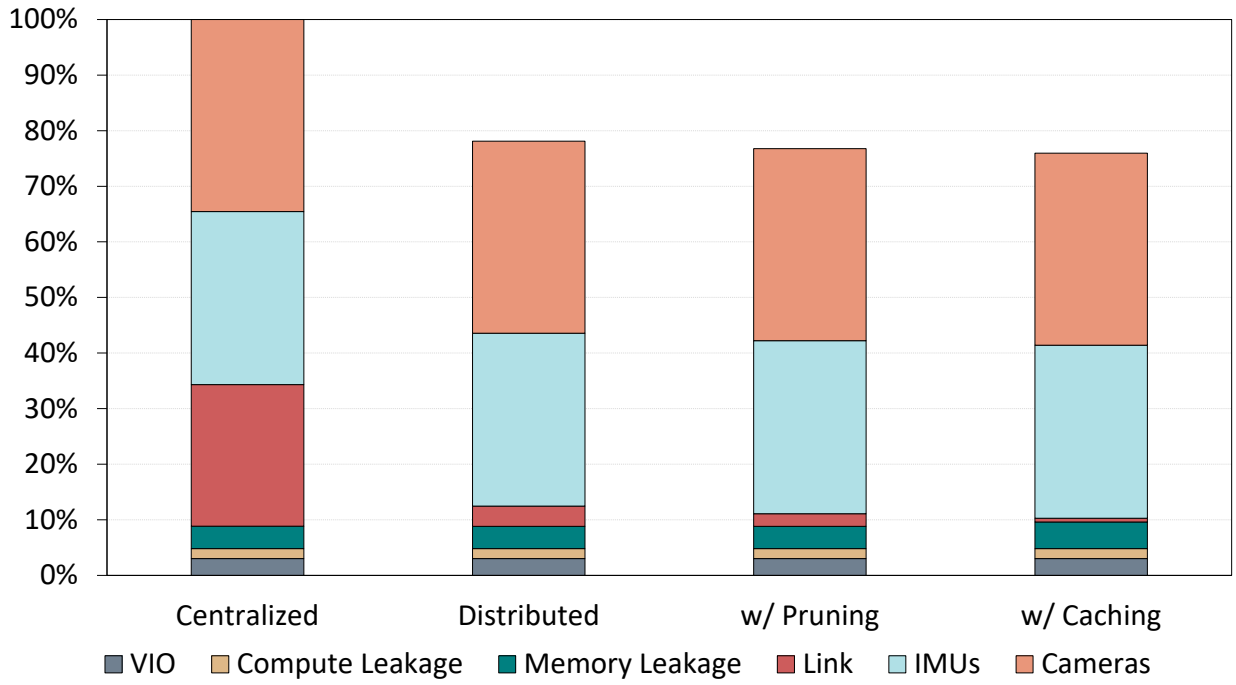


Figure 5.10: Normalized power breakdown of the entire VIO subsystem for the four VIO designs. All bars are normalized to the VIO subsystem power of the baseline centralized design.

5.5 SUMMARY

In this chapter, we presented an on-sensor distributed architecture for Visual-Inertial Odometry that reduces off-chip communication traffic by 37×, resulting in a VIO chip power reduction of 3.3×, and a total VIO subsystem power reduction of 1.3×. To the best of our knowledge, this is the first work that motivates on-sensor computing for applications such as VIO and SLAM by highlighting the crucial role of the off-chip communication links. It is also the first work to show a distributed hardware architecture for VIO.

The design principle presented in this chapter, distributed on-sensor computing, is applicable to other parts of the XR runtime as well. For instance, after this work was performed, on-sensor designs for both eye tracking [14] and hand tracking [15] were developed by others. Although these designs do not distribute work across sensors as we do, their development still highlights the efficacy of on-sensor computing in optimizing several XR components. In the future, a methodology could be developed to apply this technique more systematically and across components. Moreover, both the security and privacy aspects of on-sensor computing could be studied, as well as the impact on sensor and wire placement within the wearable device.

CHAPTER 6: ONLINE FREQUENCY CONTROL

As shown in the previous chapter, on-sensor computing is an effective design principle for reducing I/O power in XR systems. Another principle for reducing I/O power is online frequency control, which dynamically chooses the execution frequency of a component to reduce power consumption while maintaining an acceptable QoE. In cases where the component is driven by a sensor, the frequency of the sensor can also be modulated, which reduces the power consumption of both the sensor and the I/O between the sensor and the SoC. Thus, online frequency control can reduce power even more than on-sensor computing, albeit without providing the potential security and privacy benefits of on-sensor computing.

In this chapter, we apply online frequency control to reduce the power consumption of scene reconstruction, a key enabling technology for XR, and specifically for augmented and mixed reality. As described in Chapter 2, scene reconstruction builds a 3D model of the user's surroundings, which then allows virtual objects to interact with physical ones by virtue of accurate occlusion, collisions, and other physics-based effects. Scene reconstruction has seen significant development in the past decade [80, 84, 179, 180, 181, 182], with innumerable papers inspired by the seminal KinectFusion system [81]. While some algorithms can run on mobile devices [80, 183], they still consume a significant amount of power[10], making scene reconstruction an appropriate candidate for demonstrating power reduction via online frequency control.

The motivation for reducing execution frequency in scene reconstruction comes from our observation that existing algorithms all operate at a high frequency, typically 30 Hz, regardless of frame or sequence. Figure 6.1 compares the meshes obtained by running scene reconstruction at 30 Hz, 5 Hz, and 2 Hz for two different sequences. In the first sequence, it can be seen that running at 2 Hz produces a mesh that is of reasonable quality compared to the one produced by running at 30 Hz, the frequency at which depth images are fused into the model in existing systems. Since average power scales linearly with frequency ($average\ power = energy \times frequency$), reducing frequency by 15× directly reduces average power by 15×. At the same time, in the second sequence, the 2 Hz mesh is both less complete and noisier than the 30 Hz mesh. This shows that while the opportunity to reduce power by running fusion at a frequency lower than 30 Hz exists, the exact frequency varies from sequence to sequence. Thus, choosing a constant high frequency across all sequences would unnecessarily increase power consumption in some sequences, whereas choosing a low

[10]In this thesis, we opt for classical scene reconstruction approaches instead of neural-based approaches as neural-based approaches have high resource requirements, slow execution speed, and do not always surpass classical approaches. We perform a more detailed comparison in Chapter 8.

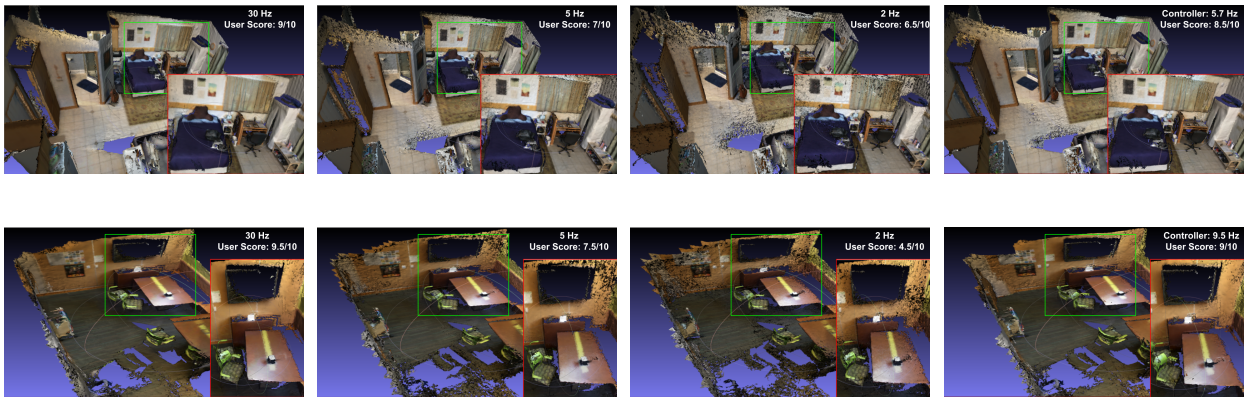


Figure 6.1: Comparison between meshes generated with various constant and dynamic scene reconstruction frequencies for two different scenes. When running at a constant 30 Hz (the commonly chosen frequency for scene reconstruction), the reconstructed mesh is good in both scenes (left). 5 Hz produces an acceptable mesh in both cases as well (center left). 2 Hz produces an acceptable mesh for the first scene but not the second one, showcasing that the minimum acceptable frequency for scene reconstruction varies from scene to scene (center right). We develop AdaptiveFusion, an online frequency controller that adapts to each scene and generates a reasonable mesh in both cases, resulting in different average frequencies (right).

frequency would degrade mesh quality in other sequences.

To pick a reasonable frequency for each sequence and to adapt to different sequences, we propose AdaptiveFusion, an online controller that varies the frequency of fusion over time in order to achieve a balance between power consumption and mesh quality. AdaptiveFusion achieves this by employing a PID controller that uses the number of newly allocated voxels per frame to make decisions. By doing so, AdaptiveFusion can increase the frequency when the user moves to a new area (which results in an increase in voxel allocations), and reduce the frequency when an area has already been mapped (which results in a decrease in voxel allocations). Our proposed system reduces fusion frequency, and thus power, by 4.9× on average, while still giving acceptable quality meshes, compared to a baseline state-of-the-art scene reconstruction implementation.

Online frequency control can be extended to other parts of the XR system, such as head tracking and the visual pipeline. To do so systematically, we develop a scheduling framework called Catan [19, 20] that determines execution frequencies of components given hardware and QoE constraints. We summarize Catan in Section 9.4.

6.1 BACKGROUND

6.1.1 Scene Reconstruction

Scene reconstruction takes a set of depth and optionally color images as input, and constructs a 3D model of the world. It enables both AR and MR applications by providing correct depth cues, occlusion for virtual objects, physics simulations, etc., and is thus an integral part of modern and future XR systems. KinectFusion [81] was the first work that supported real-time scene reconstruction using commodity depth sensors. Since KinectFusion, a plethora of works have explored RGB-D-based scene reconstruction over the past decade. Notable among these are methods that improve the scalability of scene reconstruction by using efficient data structures such as octrees [180], hash tables [179], and deformation graphs [182]. Regardless of these differences, all of these methods run at a constant 30 Hz in all cases, and do not consider dynamically changing fusion frequency based on the scene.

Some methods represent the 3D model using a volume, using unit cubes called voxels (volume elements), while others represent it using surfaces, using discs called surfels (surface elements). In this chapter, we assume a voxel-based algorithm. While surfel-based scene reconstruction algorithms have their advantages [182], voxel-based algorithms are more suitable for XR because they provide contiguous geometry, enabling easy collision checks, and are easier to extract meshes from [84].

Scene reconstruction algorithms work with both ICP-based (Iterative Closest Point) [80, 184] and VI-SLAM-based [84, 85, 183] pose trackers. We assume a non-ICP-based pose tracker in this chapter[11], and focus only on the scene reconstruction step, which consists of bilateral filtering, visibility check, and voxel fusion (Section 4.2). First, bilateral filtering reduces noise and removes outliers from the incoming depth image. Next, visibility check generates a list of voxels which are visible from the current viewing frustum. Finally, voxel fusion goes through each visible voxel and fuses depth readings into it.

6.1.2 PID Control

A proportional-integral-derivative (PID) controller is a widely-used control loop mechanism. Common examples include thermostats, cruise control in cars, and valve controllers in industrial settings. A generic PID controller aims to achieve a target value, called a setpoint, by observing a process variable and making control decisions accordingly. It does so by cal-

[11] Pose estimation is usually studied and optimized separately, as shown in Chapter 5 and in prior work [168, 170, 185].

culating the error between the setpoint and the measured process variable, and applying a correction based on proportional, integral, and derivative terms and their weights [186]. The proportional part multiplies the calculated error with a fixed proportion, aiming to reduce error quickly at the beginning and gradually slowing down as the setpoint approaches. A proportional controller cannot maintain a steady-state as it requires a non-zero error term to operate. The integral part maintains steady-state by summing up error information over time. Once steady-state is achieved, the integral output will not change since the input of the integrator (the error term) is zero. Lastly, the integrator may overshoot the setpoint depending upon historic error values. To mitigate this, the derivative part measures the rate of change of the error. If the error decreases too fast, it produces a negative offset to prevent the PID controller from overshooting the setpoint. PID controllers have been applied to SLAM in the past [187, 188], but not to change the frequency of the computation.

6.2 ADAPTIVEFUSION

Scene reconstruction typically runs at 30 Hz [80, 81]. This frequency is reasonable in systems that use ICP-based pose trackers because ICP assumes a small pose delta between the two frames being compared, necessitating a high frequency for the entire scene reconstruction pipeline. In a system with a pose estimator that is external to scene reconstruction, however, this restriction no longer applies, and bilateral filtering, visibility check, and voxel fusion—referred to simply as fusion from here on out—can potentially be performed at a lower frequency, reducing system power and hardware utilization. Even though systems exist that use an external pose tracker with scene reconstruction [84, 85], they still operate fusion at 30 Hz, and do not explore running it at a lower frequency.

However, as shown in Figure 6.1, the main challenge with choosing a lower frequency is that the optimal fusion frequency depends on both the scene and the user's motion. Thus, determining an optimal, constant frequency *a priori* is not feasible. This motivates AdaptiveFusion, a mechanism for calculating the frequency of fusion in real time. AdaptiveFusion uses a PID controller specialized for scene reconstruction to calculate the frequency of fusion.

They key insight behind AdaptiveFusion is the choice of process variable. We monitor the number of newly allocated bricks[12] in a frame to determine the instantaneous frequency of fusion; i.e., the inverse of the time until the next frame is acquired. The intuition behind using the number of newly allocated bricks as a process variable is that we only need to fuse

[12] A block of 8×8×8 voxels, the unit in which the map is allocated and managed by contemporary voxel-based scene reconstruction algorithms.

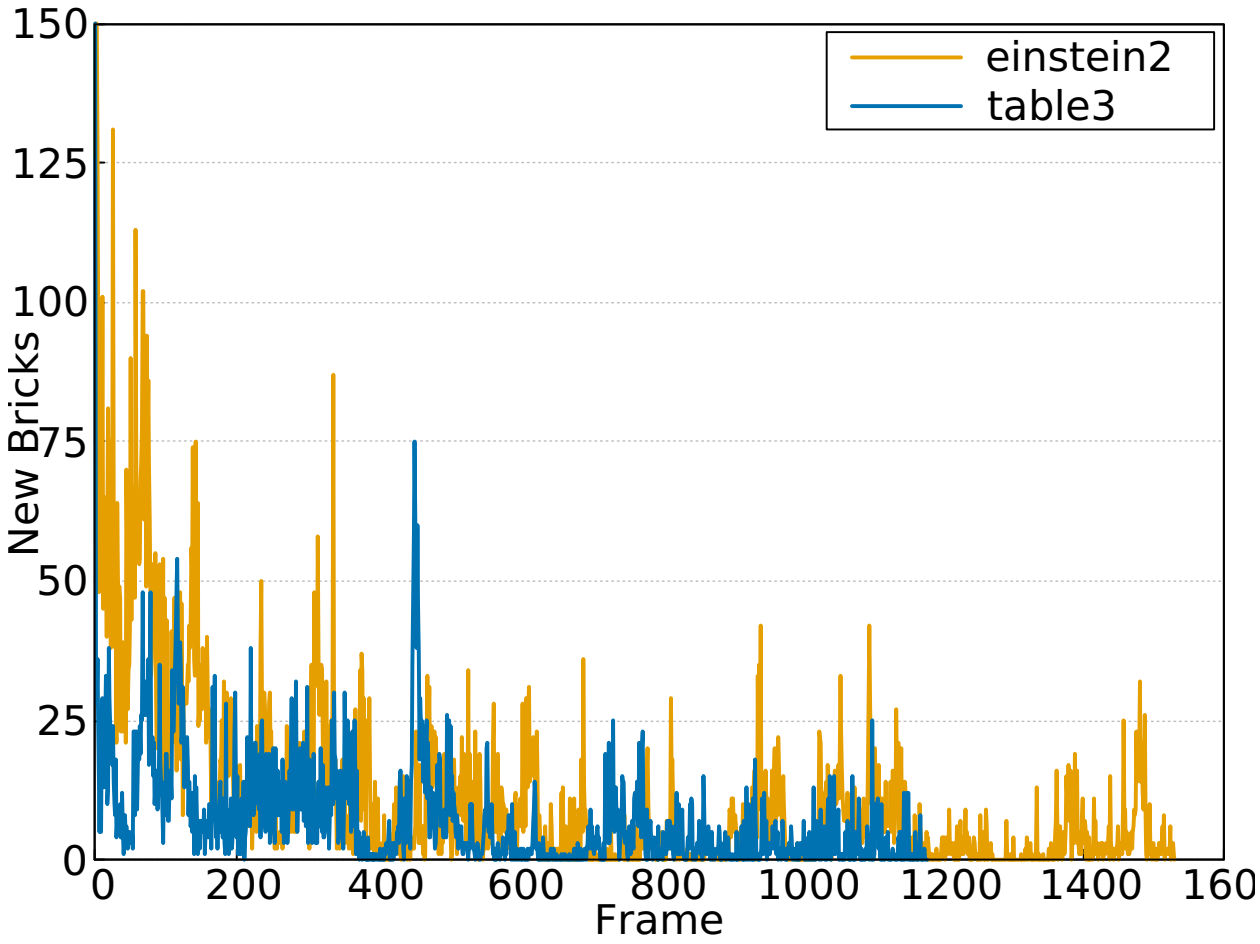


Figure 6.2: Number of new brick allocations per frame for `einstein2` and `table3` from ETH3D [189]. Note both the frame-to-frame variability and the sequence-to-sequence variability. Our controller adapts to this variability by using dynamic setpoints.

new depth data into the voxel volume at a high rate when we observe a new part of the scene. If the user is moving around in an area that has already been mapped, few bricks will be allocated, and fusion can be run at a low frequency. Conversely, when the user moves to a new area, more bricks are allocated, and fusion has to be run at a high frequency both so that new parts of the scene are not missed, and so that the new voxels can achieve high confidence (high fusion count) quickly and give an updated map to the user in a timely manner. Prior work on PID control for SLAM [187, 188] uses pose distance as the process variable, which is better suited for tracking rather than scene reconstruction.

As shown in Figure 6.2, the number of newly allocated bricks can vary significantly from frame to frame and from sequence to sequence. To make the controller adapt to these changes, we use two dynamic setpoints instead of a static one to perform PID control, similar to [188]. After each frame is executed, the minimum, maximum, and average number of

newly allocated bricks per frame, denoted by B_{min}, B_{max}, and B_{avg}, respectively, is updated. Then, the two dynamic setpoints are calculated as follows:

$$B_{low} = (1 - \alpha) \times B_{min} + \alpha \times B_{avg} \tag{6.1}$$

$$B_{high} = (1 - \alpha) \times B_{avg} + \alpha \times B_{max} \tag{6.2}$$

Figure 6.3 summarizes the decision-making process of our controller. When the number of newly allocated bricks is less than B_{low}, the lowest possible frequency setting is chosen. When the number of newly allocated bricks is more than B_{high}, the highest possible frequency setting is chosen. When the number of newly allocated bricks lies between B_{low} and B_{high}, the frequency is determined proportionally based on the difference from B_{low}. The range from B_{low} to B_{high} is divided into an equal number of frequency bins, and the bin in which the number of newly allocated bricks falls determines the frequency setting. This is the output of the P part of the controller.

The I part of the controller keeps a sliding window of the difference between newly allocated bricks and the corresponding B_{low}, and then computes another P response described above using the average value of the window. The P output is then combined with the I output.

The D part of the controller monitors the sliding window and checks whether the number of brick allocations is constantly increasing. If it is, the frequency setting is increased by one step. In practice, this part of the controller is never triggered as the number of newly allocated bricks never rises consistently. To be conservative, a constantly decreasing number of brick allocations does not result in a lower frequency setting.

The parameter α controls whether the two dynamic setpoints skew towards lower values or higher values. The lower the α, the lower the dynamic setpoints, and the higher the frequencies chosen by the controller. The higher the α, the higher the dynamic setpoints, and the lower the frequencies chosen by the controller. Additionally, constants K_p, K_i, and K_d determine the relative importance of the P, I, and D responses, respectively.

Finally, to initialize the controller, we have a bootstrap phase of 30 frames in which the sliding window is populated and the two dynamic setpoints achieve stable values. During this phase, the system runs at maximum frequency. Once the bootstrap phase is finished, the controller calculates the frequency of the system as described above.

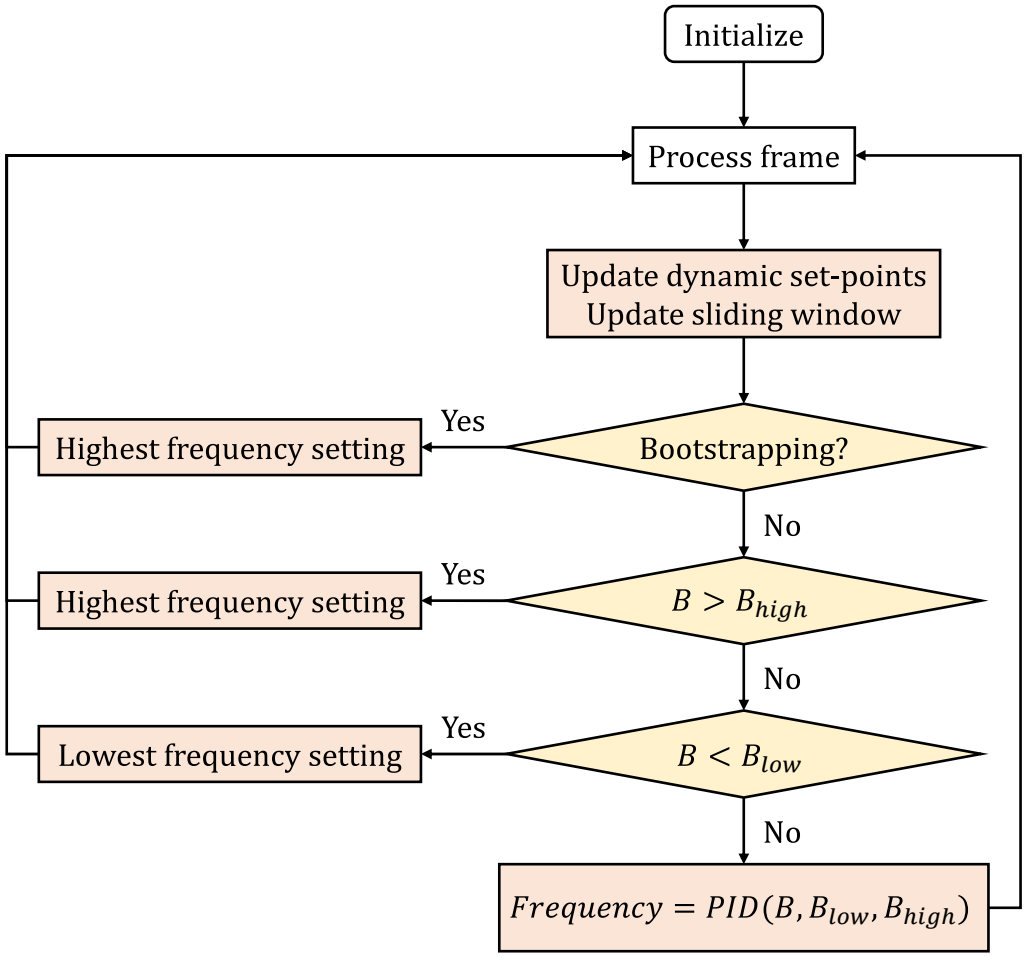


Figure 6.3: Decision making process of the frequency controller. Control flow is shown in light gold, and state updates are shown in light orange. B, B_{low}, and B_{high} denote the number of newly allocated bricks in the current frame, the low setpoint, and the high setpoint, respectively.

6.3 EXPERIMENTAL SETUP

We conduct extensive experiments over several sequences to test the efficiency of our proposed system. This section describes our experimental setup.

6.3.1 Implementation

We implement our system in the open-source InfiniTAM v3 framework [80]. InfiniTAM provides separate modular components for tracking, bilateral filtering, visibility check, and voxel fusion. Since we focus on scene reconstruction, to isolate reconstruction errors from pose estimation errors, we replace the tracking module with ground truth poses. For fre-

Table 6.1: Sequences from ScanNet used to tune the PID controller constants. Each sequence consists of depth images with a resolution of 640 × 480 pixels and color images with a resolution of 1296 × 968 pixels, both at 30 Hz.

Sequence	scene0000	scene0002	scene0004	scene0005	scene0010	scene0101
Length	5578	5193	929	1159	2514	1408

Table 6.2: Sequences from ScanNet used in the testing set. Each sequence has the same image resolution and frame rate as the sequences in the training set.

Sequence	scene0054	scene0233	scene0331	scene0465	scene0613
Length	6629	8268	6030	6307	6001

Table 6.3: Sequences from ETH3D used in the testing set. Each sequence consists of depth and color images with a resolution of 739 × 458 pixels, both at 27.1 Hz.

Sequence	desk-changing1	einstein	planar3	plant-scene1
Length	3931	1530	901	740
Sequence	repetitive	sfm-bench	sofa1	table3
Length	1965	660	976	1180

quency reduction and the online controller, we make no changes to the core algorithm, and simply control the input to the pipeline. We assume a software-triggered camera, which means camera frequency (and thus camera and I/O power) is also determined by AdaptiveFusion. In optimized configurations where fusion is run at a frequency reduced by a factor X, our system will invoke fusion only for every Xth frame, saving compute, sensor, and I/O energy on the $X - 1$ previous frames.

6.3.2 Sequences

We use several sequences from ScanNet [161] and ETH3D [189], two popular scene reconstruction and SLAM datasets, to evaluate our system, listed in Table 6.1, Table 6.2, and Table 6.3. The sequences cover a variety of scenes, camera trajectories, and difficulties, and range in length from 740 to 8268 frames (27 seconds to 4.5 minutes). We manually tune AdaptiveFusion using a training set consisting of the ScanNet sequences listed in Table 6.1. We use ETH3D and the remaining ScanNet sequences as the testing set.

6.3.3 Metrics

To evaluate the quality of the reconstructed scenes, we use F-score as defined by [190]. We extract a mesh from the final voxel volume using marching cubes, and in line with prior work [161], post-process it by merging nearby vertices, removing connected components with fewer than 7500 triangles, running two rounds of quadric edge collapse decimation, and smoothing the surface normals. We obtain ground truth meshes by passing the ground truth poses of each sequence to InfiniTAM at 30 Hz. We assume that power consumption scales linearly with frequency.

6.3.4 Configurations

We use the following frequencies to evaluate each sequence: 30 Hz, 15 Hz, 10 Hz, 7.5 Hz, 5 Hz, 2 Hz, and AdaptiveFusion. For AdaptiveFusion, we use $\alpha = 0.2$, $K_p = 0.4$, $K_i = 2.9$, and $K_d = 1.0$.

6.4 RESULTS

We first evaluate our design both quantitatively and qualitatively in Section 6.4.1. We then demonstrate the controller's adaptability and its potential to improve user experience through an in-depth analysis of its behavior in Section 6.4.2.

6.4.1 Reconstruction Evaluation

Quantitative Results Figures 6.4, 6.5, and 6.6 show the F-score of each constant frequency configuration and AdaptiveFusion for the ScanNet training set, ScanNet test set, and ETH3D, respectively. Each figure shows the F-score with both a 2 cm error threshold and a 5 cm error threshold (representing approximately 0.4% and 1% of the scene's scale). The F-scores are computed by taking the 30 Hz mesh as the reference. We cap the range of F-scores values on the y-axis for readability, and we use a log scale on the x-axis to spread the data points evenly. We make the following observations from these results.

First, for a given target F-score, the minimum frequency that meets that score varies significantly from sequence to sequence. For instance, the acceptable frequency range is from 4 Hz to 16 Hz if an F-score of 0.98 (5 cm threshold) is chosen for the ScanNet training set in Figure 6.4. Second, AdaptiveFusion (denoted using star markers) is able to achieve a good trade-off between mesh quality and frequency, selecting unique frequencies for each

Table 6.4: Minimum frequencies required to obtain minimally acceptable and acceptable meshes (based on visual inspection) for each sequence. *Min* denotes minimum acceptable frequency, *Acc* denotes a more acceptable frequency, and *Ctrl* denotes the average frequency attained by AdaptiveFusion. The range of acceptable frequencies across sequences is wide, and AdaptiveFusion is able to find a sweet spot in most cases. Note that the average frequency of AdaptiveFusion cannot be directly compared to an equivalent constant frequency due to the different instantaneous behavior of the two systems.

	Min / Acc / Ctrl		Min / Acc / Ctrl
scene0000	5 / 7.5 / 5.7	scene0613	2 / 5 / 4.2
scene0002	5 / 7.5 / 3.9	ETH3D	
scene0004	7.5 / 10 / 13.9	desk-changing1	2 / 7.5 /3.1
scene0005	2 / 5 / 9.5	einstein2	7.5 / 15 / 3.6
scene0010	5 / 7.5 / 8.5	planar3	2 / 5 / 4.4
scene0054	2 / 5 / 8.4	plant-scene1	2 / 5 / 4.4
scene0101	10 / 15 / 8.8	repetitive	2 / 5 / 5.1
scene0233	15 / 30 / 4.2	sfm-bench	7.5 / 10 / 5.4
scene0331	2 / 5 / 6.6	sofa1	2 / 2 / 7.9
scene0465	15 / 30 / 5.2	table3	15 / 30 / 6.3

sequence and generating meshes with acceptable quality. In some sequences, the star marker is even higher than the constant frequency trend line, meaning that the PID controller can either achieve similar mesh quality by fusing fewer frames or fuse the same number of frames and produce a better mesh. We find that these results hold with both error thresholds. Finally, AdaptiveFusion generalizes well, and is able to produce good quality meshes in the test sequences as well; e.g., the star markers are above the constant frequency trend line for some sequences in both Figures 6.5 and 6.6.

Qualitative Results Choosing a target F-score is not trivial, as the score curve changes drastically with the chosen error threshold. Furthermore, while F-scores correctly capture the general relationship between mesh quality and frequency, they do not convey any information about the perceptual quality of a mesh. Therefore, we also visually inspect all the meshes. Figure 6.7 shows the reconstructed meshes for four sequences from ScanNet, each with four configurations: constant 30 Hz (best achievable mesh), the constant frequency that generates a minimally acceptable mesh, the constant frequency that generates a more acceptable mesh, and AdaptiveFusion, in that order. The frequencies are chosen based on visual inspection of the meshes, and are listed for all sequences in Table 6.4.

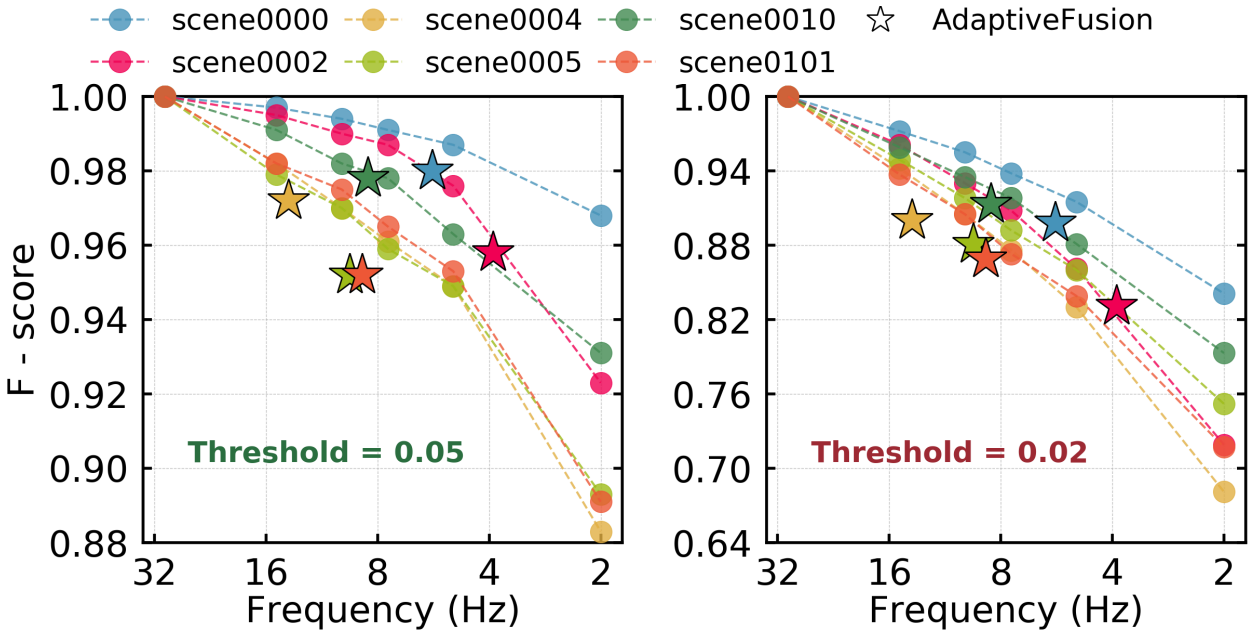


Figure 6.4: F-scores with 5 cm and 2 cm thresholds for the ScanNet training set.

Both Table 6.4 and Figure 6.7 corroborate the F-score results, with different sequences exhibiting different optimal frequencies, and AdaptiveFusion successfully adapting to each scene. In each case, AdaptiveFusion generates a mesh that is better than the minimally acceptable mesh and comparable to the more acceptable mesh. For `scene0010` and `scene0331`, AdaptiveFusion achieves an average frequency that is slightly higher than required. However, even when its achieved average frequency is lower than the desired frequency (`scene0002` and `scene0101`), the quality of the generated mesh is comparable or better. This is because the average frequency of AdaptiveFusion cannot be directly compared to an equivalent constant frequency, as AdaptiveFusion fuses frames based on scene information, which sometimes results in better meshes with fewer fusions.

The qualitative results also show that there is a need for mesh quality assessment metrics that better align with subjective quality scores. While F-score reflects most mesh quality changes (higher qualities corresponds to higher F-scores), it is still not a perfect fit for evaluating scene reconstruction quality. For example, in `scene0010`, AdaptiveFusion's mesh has a worse F-score than 7.5 Hz's mesh, despite the AdaptiveFusion mesh being visually better than the mesh generated by 7.5 Hz. We hope these results motivate researchers to develop better perceptual mesh quality metrics.

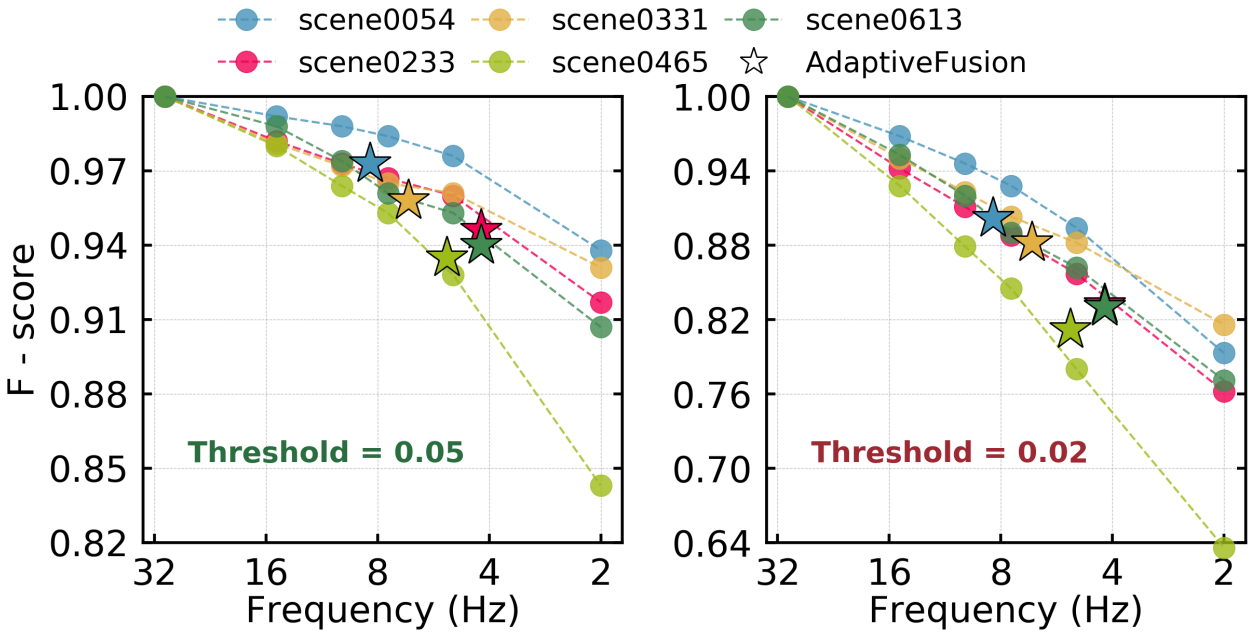


Figure 6.5: F-scores with 5 cm and 2 cm thresholds for the ScanNet testing set.

6.4.2 PID Controller Analysis

Controller Activity AdaptiveFusion monitors the number of newly allocated bricks per frame to determine the instantaneous frequency of fusion. To both understand how the controller's decisions are affected by different characteristics of different sequences, and its ability to generalize from the training to the test set, we analyze the controller's activity in two representative data sequences: `scene0002`, a training sequence with several large objects in the center of the room, and `scene0233`, a test sequence with furniture closer to the walls.

Figure 6.8 shows the variation of newly allocated bricks and the controller's output (top and bottom, respectively) over time for the two aforementioned sequences. Each spike in brick allocations corresponds to a region becoming visible for the first time. In both sequences, the controller responds to spikes in brick allocation appropriately, although the exact frequency that it chooses vary greatly based on both the magnitude of the spike and the brick allocation history till that point. In `scene0002`, there are few allocations once half the sequence has passed, and the controller responds by running at a very low frequency. In contrast, `scene0233` has a steady stream of brick allocations throughout the sequence, resulting in the controller choosing higher frequencies. Overall, the controller is able to find a balance between fusing quickly when new areas are discovered, and fusing slowly when the trajectory traverses an already mapped area.

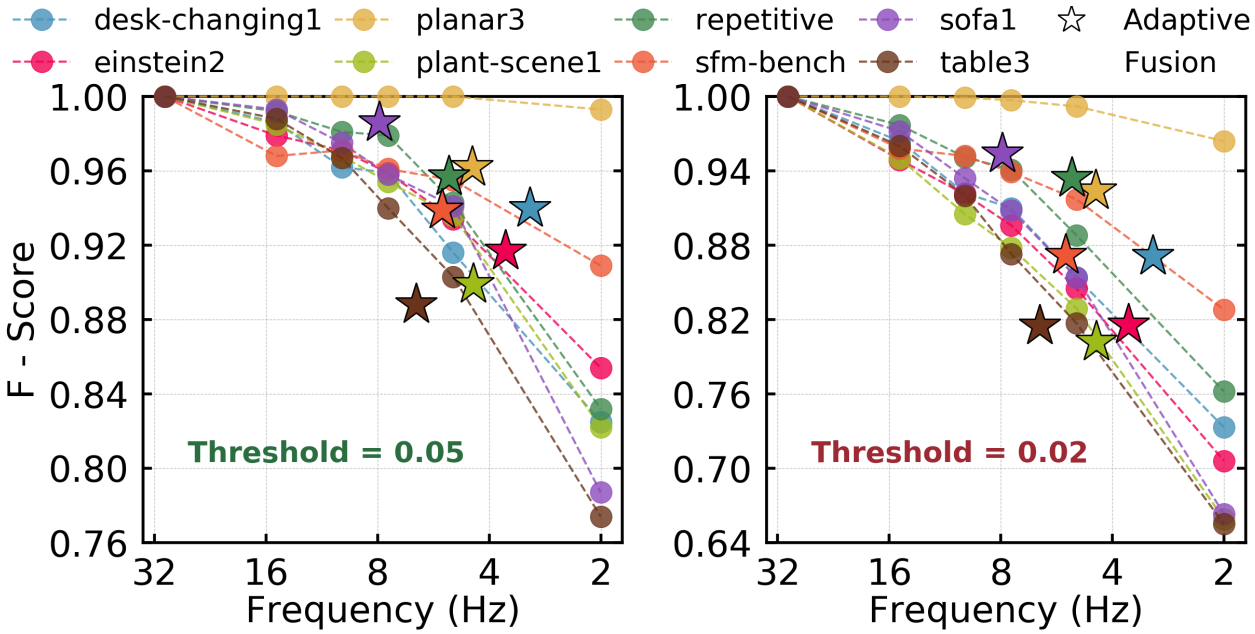


Figure 6.6: F-scores with 5 cm and 2 cm thresholds for the ETH3D testing set.

Motion-to-Mesh Latency The responsiveness of our controller brings an additional benefit in the form of reduced *Motion-to-Mesh Latency* (MTM). Similar to Motion-to-Photon latency, a widely adopted metric in the XR community, we define a new metric, MTM, as the period from a user seeing a new scene to the scene getting reasonably reconstructed and becoming ready for use. While we can fuse at a constant rate, either high or low, neither option can balance mesh quality and power consumption: a higher fusing rate reduces MTM but also incurs a higher power cost. On the other hand, a lower fusing rate saves more power but compromises user experience due to a high MTM.

Figure 6.8 shows that our controller reacts appropriately to newly visible parts of the scene: it sustains a high frequency for a brief period of time to quickly obtain a reasonable reconstruction, and then immediately returns to a low frequency state to save power. Consequently, it is able to minimize both MTM and power consumption.

6.5 SUMMARY

In this chapter, we showcased online frequency control by proposing AdaptiveFusion, the first work that demonstrates that it is possible to dynamically choose scene reconstruction frequency by observing the number of newly allocated bricks per frame. Compared to an existing state-of-the-art baseline, AdaptiveFusion's PID controller reduces power consumption by 4.9× while maintaining acceptable mesh quality.

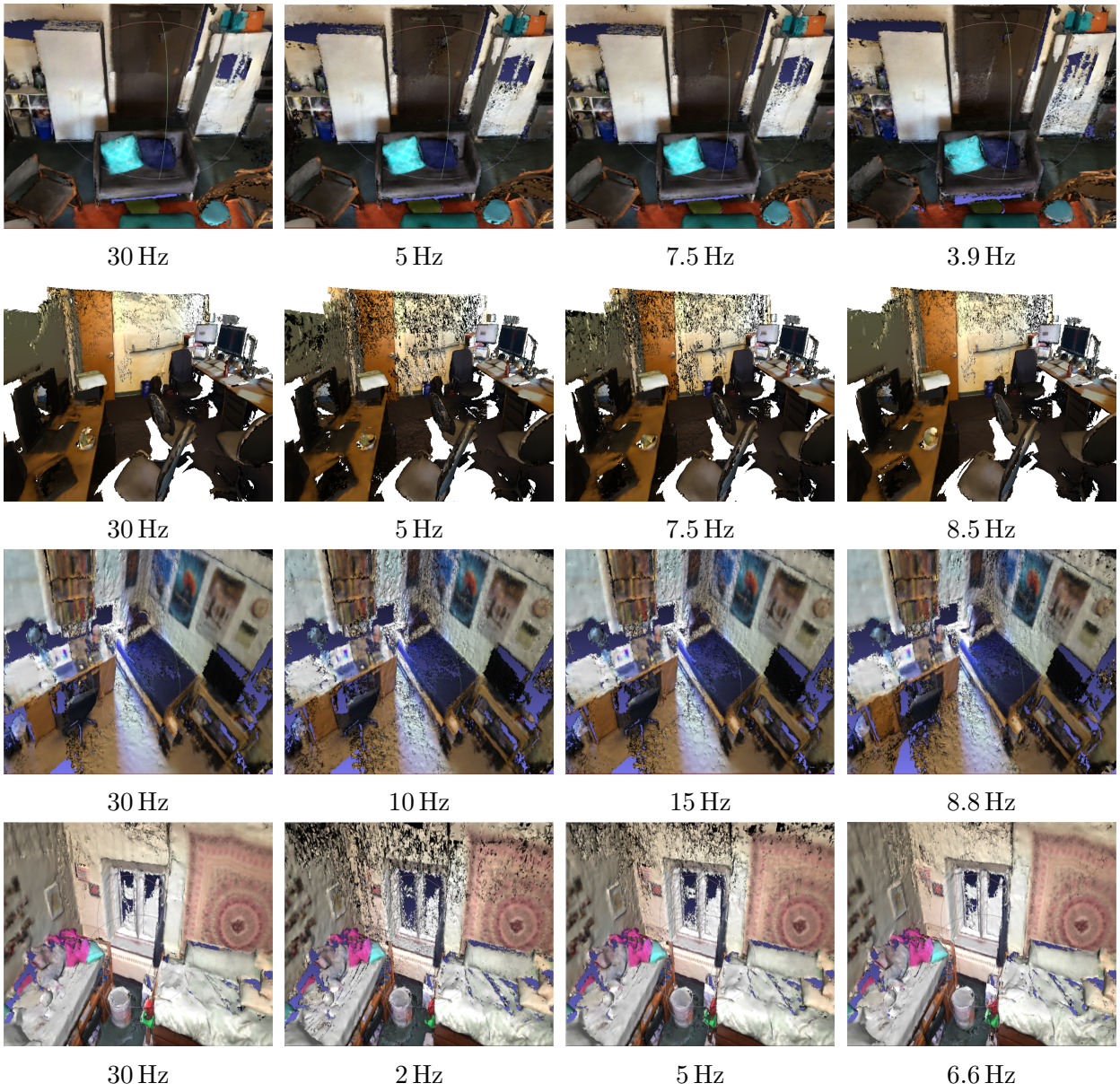
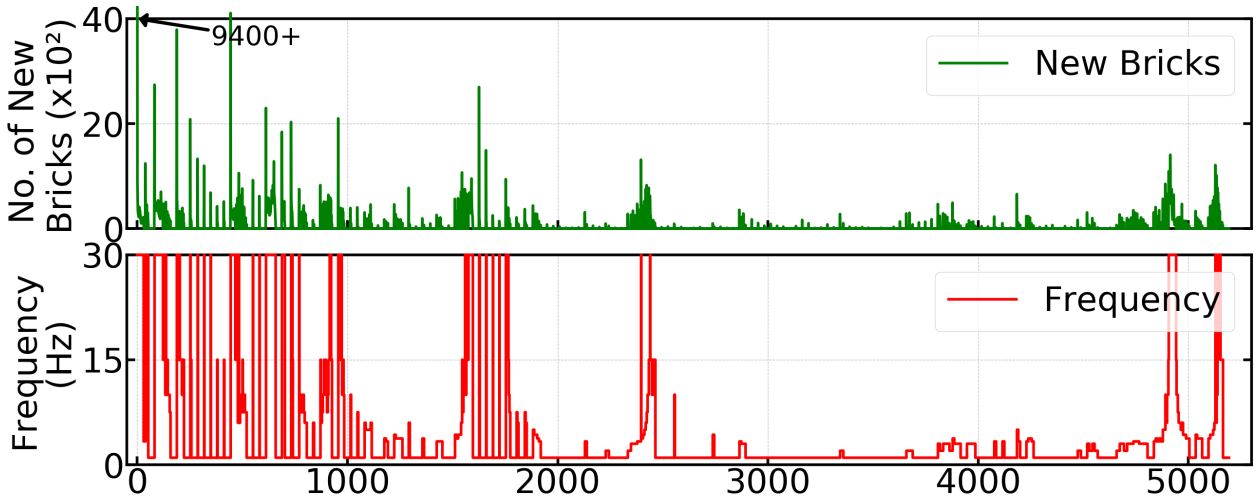
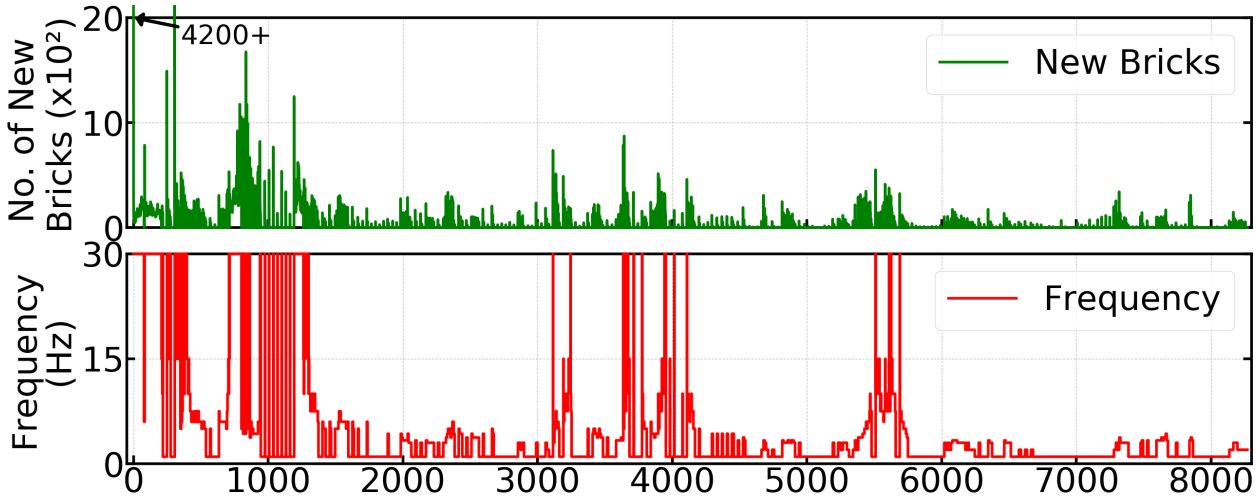


Figure 6.7: We compare meshes generated by AdaptiveFusion to meshes generated by running at various constant frequencies for ScanNet scenes 2, 10, 101, and 331. For each scene, the first column shows the mesh generated by running at the maximum possible frequency, 30 Hz. The second column shows the minimally acceptable mesh and the corresponding constant frequency. The third column shows a more acceptable mesh and its corresponding constant frequency. The last column shows the mesh generated by AdaptiveFusion and its average frequency for the scene. AdaptiveFusion-generated meshes are on par with and sometimes better than constant frequency meshes.

At the same time, AdaptiveFusion overshoots the optimal frequency in some cases, and undershoots it in others. Furthermore, the response of the PID controller can sometimes



(a) `Scene0002`. A living room with furniture in the center.



(b) `Scene0233`. A dormitory with furniture near the walls.

Figure 6.8: Newly allocated bricks and output of the frequency controller over time. The y-axis of newly allocated bricks is capped at 4000 and 2000 , respectively, for readability. The red peaks that intersect with the figure's upper boundary are all 30 Hz.

fluctuate between extremes, potentially resulting in undesirable instantaneous power and thermal fluctuations. Addressing these shortcomings is a good avenue for future research.

Our results also show that quantitative metrics such as F-score do not always match the subjective experience of human beings, who are ultimately the final arbiters of what is and is not an acceptable mesh. Furthermore, the target F-score for a given sequence is not immediately obvious. These shortcomings of existing quantitative metrics motivate research in human perception-driven metrics for room-scale meshes.

Finally, other parts of the XR system such as head tracking and the visual pipeline can benefit from online frequency control as well. To approach this problem methodically for the

entire system, we develop Catan [19, 20], a scheduling framework that determines component frequencies to meet QoE constraints. We summarize Catan in Section 9.4.

CHAPTER 7: SCALABLE HARDWARE MAPPING

The two design principles that we have explored thus far do not perform any hardware specialization, which is often necessary to achieve the aggressive power targets laid out in Section 2.1—1 W peak power and 100 mW average power. At the same time, it is infeasible to build a dedicated accelerator for each task of each component, as discussed in Section 4.4, both due to the changing nature of the algorithms, the additive leakage power and area overheads, and engineering costs [191]. Therefore, a more scalable approach to hardware specialization is required, one which maximizes reuse of existing hardware components before generating new ones. We call this approach scalable hardware mapping, and apply it to scene reconstruction to further reduce its power consumption.

As in the previous chapter, we focus solely on the core scene reconstruction tasks: bilateral filtering, visibility check, and voxel fusion. Guided by our analysis presented in Section 4.2, we map bilateral filtering to an Image Signal Processor (ISP), visibility check to a Digital Signal Processor (DSP), and propose a simple accelerator for voxel fusion. Our approach is both feasible and scalable, as it reuses two existing hardware components, and only adds one new one. This is in contrast with previous approaches, which either propose custom hardware for the entire component [192], or propose intrusive system-wide changes which only accelerate a small fraction of the entire component [193].

When combined with AdaptiveFusion, our proposed hardware mapping consumes less than 8 mW, 125× less than running scene reconstruction on an embedded GPU. Scalable hardware mapping is naturally applicable to other components as well. To apply it in a principled fashion, we develop a tool called Trireme [21] which automatically selects acceleration candidates. We summarize Trireme in Section 9.5.

7.1 BACKGROUND

Similar to Chapter 5, we assume a voxel-based scene reconstruction algorithm in this chapter. We do not consider pose estimation as it is usually optimized (and often accelerated in hardware) separately, as shown in both Chapter 5 and prior work on pose estimation hardware [168, 170, 185]. Thus, we focus on the core scene reconstruction tasks: bilateral filtering, visibility check, and voxel fusion.

First, bilateral filtering reduces noise and removes outliers from the incoming depth image. A bilateral filter replaces the intensity of each pixel with a weighted average of the intensities of its neighbors. Next, visibility check generates a list of voxel bricks which are visible from

the current viewing frustum. This is achieved by projecting each pixel in the depth image into 3D space, and marking the brick it falls upon as visible. Since depth images are noisy, to be conservative, all voxel bricks along a small ray around the depth point are marked as visible. Finally, voxel fusion goes through each voxel inside each visible brick, projects the voxel into the depth image, and fuses the depth reading into the voxel if the voxel lies within the truncation band of the depth point. A detailed overview of these tasks can be found in KinectFusion [81] and InfiniTAM [80].

7.2 HARDWARE MAPPING

Before accelerating scene reconstruction in hardware, a natural question to ask is whether running scene reconstruction on a GPU will suffice, especially since some existing scene reconstruction algorithms run in real-time on mobile GPUs [80, 183]. In XR systems, running scene reconstruction on the GPU is not feasible due to two reasons.

First, despite their efficiency at executing parallel workloads, GPUs consume a significant amount of active and idle power. As an example, we ran AdaptiveFusion on an NVIDIA Jetson Xavier development board [126], and found that the GPU consumed upwards of 1 W, despite AdaptiveFusion significantly reducing the execution frequency. We also observed that the Dynamic Voltage and Frequency Scaling (DVFS) mechanism only began to lower voltage and frequency after the GPU was idle for several seconds. Since such long idle periods rarely occur in practice, idle power remained high. Thus, meeting the aforementioned peak and average power targets would not be possible by running scene reconstruction on the GPU.

Second, the primary purpose of a GPU in an SoC is to run graphics workloads. As such, the GPU is usually well utilized all the time, and scheduling scene reconstruction on it may not be possible. Even if GPU utilization were low, the issue of scheduling remains. For instance, asynchronous reprojection (Section 2.4.1) is a critical task which needs to run on the GPU immediately before each image is displayed, and running scene reconstruction on the GPU may interfere with it. Thus, it is desirable to run scene reconstruction on other hardware.

In order to determine the mapping of each algorithmic task to hardware, we revisit the analysis of scene reconstruction presented in Section 4.2, and obtain the following insights.

7.2.1 Compute

First, we find that bilateral filtering is a traditional image processing task, and can thus be mapped to an ISP. Second, we find that after performing a minor rewrite of the software, the only task which is affected by choice of data structure is visibility check, as it is the only task which needs to access the hash table. Thus, in order to retain algorithmic flexibility, it is *only* visibility check that has to be mapped on to programmable logic. Consequently, visibility check is well-suited for a DSP, as a DSP is both sufficiently programmable and adept at exploiting parallelism. Both ISPs and DSPs are readily available in virtually all modern mobile SoCs, and are more power-efficient than GPUs, consuming only tens of milliwatts [194, 195].

Finally, we note that voxel fusion has remained unchanged since it was proposed over 26 years ago [196], and is not likely to undergo changes in the near future either. Therefore, it is safe to build a full-custom accelerator for voxel fusion poses. Since each brick contains $8 \times 8 \times 8$ voxels, a straightforward implementation is to have an 8-wide datapath with each lane consisting of a fully pipelined functional unit that implements the TSDF projection and update equations. Once the pipeline fills up, the datapath can process 8 voxels each cycle, and a brick every 64 cycles. Additionally, since voxels are 8 B each, a single 64 B cache line can service all 8 lanes simultaneously.

7.2.2 Memory

The aforementioned mapping mitigates the need for designing custom datapaths for bilateral filtering and visibility check. We next make the case for **not** needing custom memory hierarchies, which are often a fundamental aspect of designing efficient hardware accelerators. First, since bilateral filtering is a standard stencil operation, it possesses excellent spatial and temporal locality. ISPs are already optimized for exploiting such locality, and thus no extra effort is required to optimize the execution of bilateral filtering on the ISP. Second, even though visibility check contains irregular hash table accesses, the working set size of the hash table is small, typically a few hundred kB, which can easily fit in the L2 cache of a modern DSP [197].

Finally, as described in Section 4.2, voxel fusion has streaming accesses to voxel bricks *within* a frame and 80%–90% data reuse of voxel bricks *across* frames. The streaming accesses can be easily prefetched, and inter-frame reuse can be exploited by simply having a large enough last-level cache (LLC). Since modern SoCs have 8–16 MB LLCs [198], no extra storage is required to exploit this inter-frame reuse.

7.3 EXPERIMENTAL SETUP

We text the efficacy of our proposed hardware mapping with several sequences. This section describes our experimental setup.

7.3.1 Implementation

Our proposed hardware mapping cannot be modeled on existing SoCs as these SoCs do not contain our custom voxel fusion accelerator. At the same time, it is neither feasible to build a custom SoC nor to model each hardware component in isolation, as capturing the communication between the hardware components is important for understanding the energy-efficiency of our proposed mapping. Therefore, we opt for a hybrid approach based on analytical modeling and a trace- and event-driven timing simulator.

Scene reconstruction is a highly scene-dependent component, and the characteristics of scenes vary significantly both from frame to frame and from sequence to sequence. Therefore, in order to obtain a realistic picture of the hardware's performance, hundreds of frames have to be evaluated for each sequence, which is infeasible with RTL. Therefore, instead of performing High Level Synthesis (HLS), we model our system using a trace- and event-driven timing simulator, which we implement using the open-source Sparta framework [199]. Our simulator simulates 300K cycles per second, which is approximately two orders of magnitude faster than an RTL simulation of comparable complexity.

To drive our simulator, we collect a functional trace from InfiniTAM [80]. We instrument bilateral filter, visibility check, and voxel fusion to collect the trace, and we break down the trace of each task into *work items* that can be executed independently of each other, similar to CUDA threads. A work item can be a different logical unit of work based on the algorithmic task; e.g., each pixel represents a work item in bilateral filter, and each voxel represents a work item in voxel fusion. Each work item consists of a sequence of instructions which must be executed sequentially. Our functional trace consists of two types of instructions: compute and memory. Compute instructions specify a fixed number of cycles, whereas memory instructions encode the virtual address of the memory access. We combine all instructions in between two memory instructions into one specialized compute instruction. Although branch instructions are not part of the functional trace, we use them to demarcate compute instructions in order to capture input-dependence.

We model each hardware component described in Section 7.2, including the ISP and DSP, as a *unit*. Each unit consists of a work item scheduler, one or more processing elements (PEs), and a shared L1 data cache (L1D). Each PE can have one or more concurrent resident work

items. Each cycle, the scheduler attempts to issue an instruction to each PE. When a PE receives a compute instruction, it simply notifies the scheduler of instruction completion after the number of cycles specified in the trace have elapsed. When a PE receives a memory instruction, it issues it to its L1D, where it follows the typical memory request service flow. All the units, LLC, and memory controller are connected to each other via an on-chip network (NoC), which is analytically modeled using flit size and the number of flits to be transmitted. Once a unit finishes execution, it launches the next unit in algorithm order (we do not pipeline the execution of the various units).

Although we do not build dedicated ISP and DSP models in our simulator, our modeling methodology above yields representative results due to our energy model (described below). Our bilateral filtering and visibility check units both consume of the order of 10 mW while running, which is in line with publicly available ISP [195] and DSP [194] numbers.

We model the energy consumption per frame as described by equations 7.1–7.9. We calculate the number of operations executed by a PE in terms of equivalent FMA16 operations.

$$\text{scheduler} = |\text{instructions}| \times \text{issue energy/instruction} \tag{7.1}$$

$$\text{PE} = |\text{ops}| \times \text{energy/op} + |\text{memory instructions}| \times \text{energy/lsq enqueue} \tag{7.2}$$

$$\text{L1D} = |\text{sram accesses}| \times \text{energy/access} + |\text{total accesses}| \times \text{energy/MSHR lookup} \tag{7.3}$$

$$\text{unit} = \text{scheduler} + \sum \text{PE} + \text{cache} \tag{7.4}$$

$$\text{NoC} = |\text{flits}| \times \text{energy/flit} \tag{7.5}$$

$$\text{LLC} = \text{same as L1D model} \tag{7.6}$$

$$\text{DRAM} = |\text{bits}| \times \text{energy/bit} \tag{7.7}$$

$$\text{MIPI} = |\text{bits}| \times \text{energy/bit} \tag{7.8}$$

$$\text{total} = \sum \text{unit} + \text{NoC} + \text{LLC} + \text{DRAM} + \text{MIPI} \tag{7.9}$$

7.3.2 Sequences

We use several sequences from ScanNet [161] and ETH3D [189] to evaluate our system, listed in Table 7.1. We simulate 200 frames from each sequence due to simulation time constraints.

Table 7.1: Sequences from ScanNet and ETH3D used to evaluate our proposed system. ScanNet sequences `scene0233`, `scene0331`, and `scene0465` consist of depth images with a resolution of 640×480 pixels and color images with a resolution of 1296×968 pixels, both at 30 Hz. ETH3D sequences `einstein` and `table3` consist of depth and color images with a resolution of 739×458 pixels, both at 27.1 Hz.

Sequence	`scene0233`	`scene0331`	`scene0465`	`einstein`	`table3`
Length	8268	6030	6307	1530	1180

Table 7.2: Timing parameters of the modeled hardware. We use the same number of PEs, work items per PE, and L1D for each hardware unit.

PEs	Work Items Per PE	L1D	LLC	Memory Controller	Frequency
4	32	8-way 32 kB	16-way 16 MB	2 32-bit channels	250 MHz
		1 cycle	7 cycles	10 cycles	

Table 7.3: Energy parameters of the modeled hardware. Access numbers for the L1, LLC, NoC, DRAM, and camera link are all for a 64 B cache line. All numbers are in picojoules. Due to the difficulty of obtaining a consistent set of numbers from a single source, we distill these constants from various sources (op [200, 201], SRAM [201, 202], flit [203], DRAM [201], and MIPI [177, 178]) and convert them into 5 nm numbers by scaling 2× every other node. Issue, LSQ, and MSHR energies are best effort estimates.

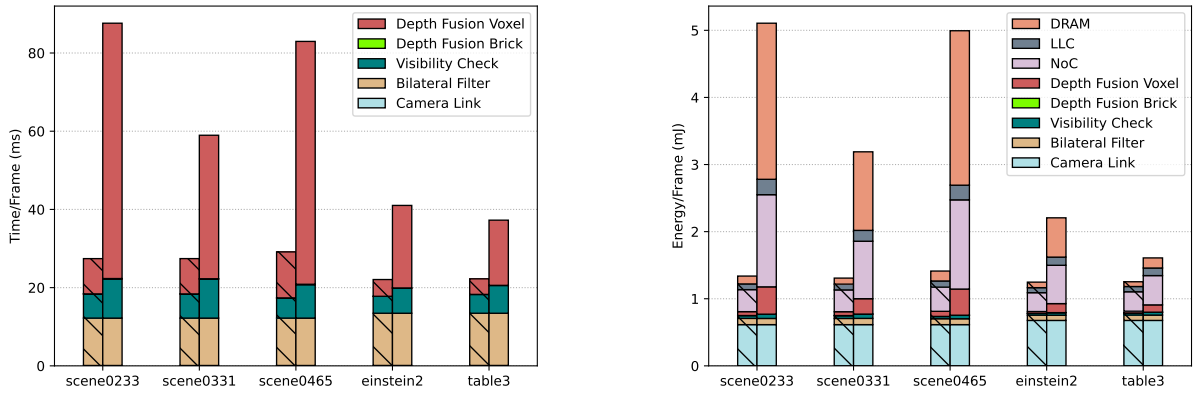
Issue	Op	LSQ	L1 Access	L1 MSHR	LLC Access	LLC MSHR	Flit	DRAM	MIPI
2	0.45	1	16	1	160	1	476	2560	25600

7.3.3 Configurations

We use AdaptiveFusion with $\alpha = 0.2$, $K_p = 0.4$, $K_i = 2.9$, and $K_d = 1.0$ to collect the functional traces. We use a truncation band of 20 mm, and a viewing frustum of [0.3, 4] m. We use both 10 mm voxels and 5 mm voxels to evaluate our design. We use 10 mm voxels as the baseline both because ScanNet uses the same size [161] and because a larger voxel size would be acceptable due to the resource constraints of XR. We use 5 mm voxels to represent aspirational XR use cases requiring the highest possible fidelity. The timing and energy parameters of our hardware design are listed in Tables 7.2 and 7.3, respectively.

7.4 RESULTS

Figure 7.1 shows the average execution time and energy per frame of our proposed hardware architecture for 200 frames of ScanNet sequences `scene0233`, `scene0331`, and `scene0465`, and ETH3D sequences `einstein2` and `table3`. For each sequence, we show the



(a) Time Per Frame

(b) Energy Per Frame

Figure 7.1: Average execution time and energy per frame breakdown for three ScanNet and two ETH3D sequences. The left bar in each cluster shows the breakdown with 10 mm voxels, and the right bar shows the breakdown with 5 mm voxels. Bilateral Filter, Visibility Check, Depth Fusion Brick, and Depth Fusion Voxel represent the compute within each unit. Note that the camera image's link transfer time is negligible and cannot be seen in the execution time graph.

time and energy with both 10 mm (left bar with hatches) and 5 mm (right bar) voxels.

7.4.1 Execution Time

Figure 7.1a shows that with 10 mm voxels, average execution time per frame remains below 30 ms for all sequences, enabling the hardware to achieve 30 Hz when required. On the other hand, using 5 mm voxels substantially increases the total amount of work to be performed, significantly increasing execution time. Given that depth fusion is the only task that accesses brick data and can exploit brick reuse across frames, it is impacted the most, and thus has the highest contribution to the increase in execution time. Visibility check does observe an increase in execution time due to the larger number of bricks that have to be processed, although the increase is not as significant as the hash table still fits in the LLC. In contrast, bilateral filter's time remains constant, as the computation is independent of voxel size and the camera image fits in the LLC in both cases. Note that the change in execution time with the two voxel sizes is highly dependent on the sequence, with `scene0233` observing the highest slowdown due to its large working set size, and `table3` observing the least due to its relatively small working set size.

7.4.2 Energy

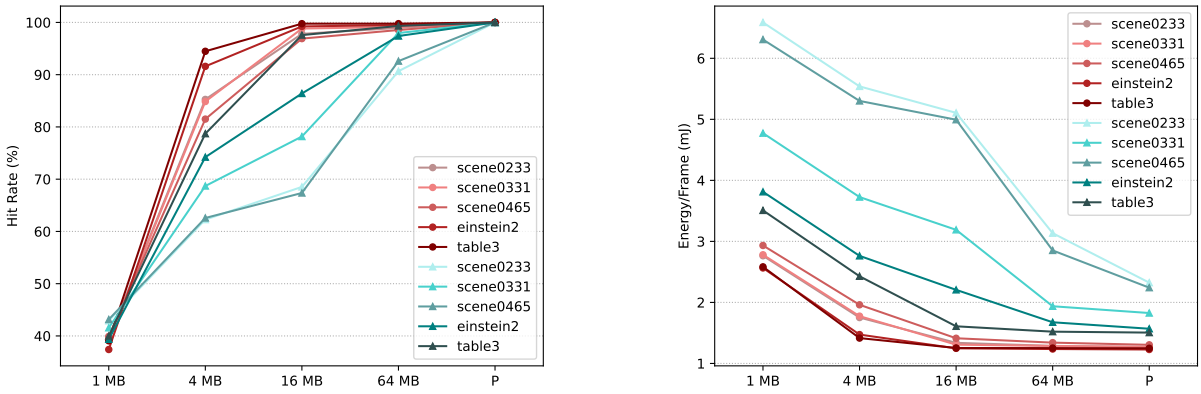
Figure 7.1b paints a similar picture to Figure 7.1a, with the 10 mm voxel configuration consuming close to 1 mJ per frame in all cases, and the 5 mm voxel configuration consuming up to 5× more than that, depending upon the sequence. However, there are three salient aspects of the energy breakdown that are not present in the execution time breakdown. First, even though transferring the camera image over MIPI CSI consumes a negligible amount of time, it does consume a significant amount of energy as MIPI CSI has an average energy per bit of 50 pJ. This is in line with our results from Chapter 5, once again showing link power to be a major component of sub-system power — in the 10 mm voxel case, the camera link consumes 40% to 50% of total frame energy.

Second, the functional units only consume 10%–20% of energy each frame, and energy is instead dominated by what is commonly known as the "uncore"—LLC, on-chip network, and memory controller and DRAM. When moving from 10 mm voxels to 5 mm voxels, it is the uncore that experiences the highest increase in energy consumption, with the NoC and DRAM being larger contributors than the LLC. The underlying cause is both the drastic increase in working set size, which increases NoC and LLC energy, and the drop in LLC hit rate, which increases DRAM energy.

Finally, regardless of sequence and voxel size, the hardware consumes less than 5 mJ per frame, and as low as 1 mJ per frame with 10 mm voxels. With the frequencies that AdaptiveFusion achieves (Section 6.4.1), the resulting average power consumption with 10 mm voxels ranges from 3.7 mW to 7.8 mW, a 128× – 270× reduction compared to running AdaptiveFusion on the GPU of an NVIDIA Jetson AGX Xavier development board, which consumes 1 W. The power consumption increases to 10–26 mW with 5 mm voxels, which is still significantly lower than the GPU baseline.

7.4.3 Cache Size Sensitivity

To better understand how voxel size impacts the behavior of our proposed hardware architecture, we perform a cache size scaling study. Figure 7.2 shows the average LLC hit rate and overall energy per frame for various LLC sizes with both 10 mm voxels and 5 mm voxels. Figure 7.2a shows that for both voxel sizes, the first inflection point occurs with a 4 MB LLC, which is able to fully cache the 1.2 MB double buffered camera image, as well the brick hash table, which consumes tens to hundreds of kB. The second inflection point occurs with a 16 MB LLC for 10 mm voxels. Using 10 mm voxels typically results in 2000 or less visible bricks per frame, which have a working set size of approximately 4 MB (4 kB per



(a) LLC Hit Rate

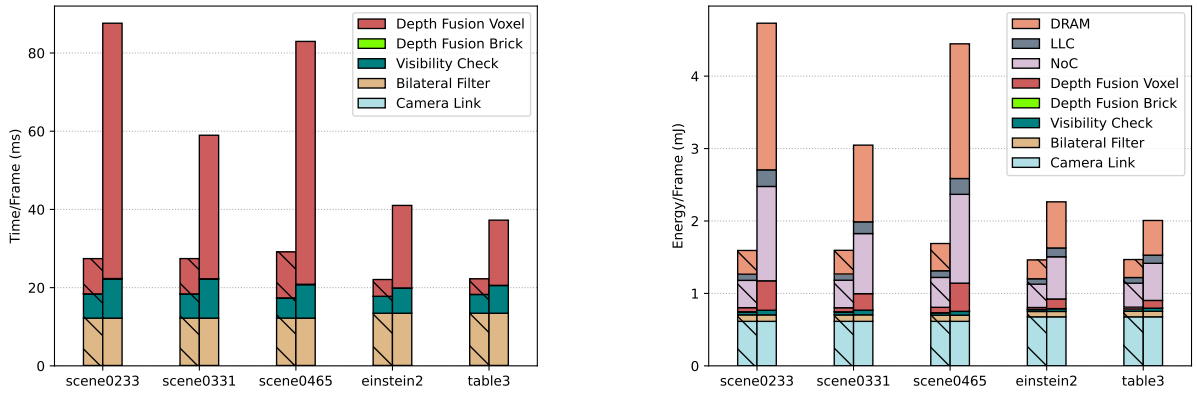
(b) Energy Per Frame

Figure 7.2: Average LLC hit rate and energy per frame for various LLC sizes. Results with 10 mm voxels are shown in shades of red with circular markers, and results with 5 mm voxels are shown in shades of blue with triangular markers. P denotes a perfect cache that always hits.

brick with approximately 50% of a brick being accessed each frame). Thus, even an 8 MB LLC could cache both the camera image and the brick data, and a 16 MB LLC is able to do so comfortably for *all* sequences.

In contrast, using 5 mm voxels typically results in tens of thousands of visible bricks each frame; i.e., a brick working set size of tens of megabytes, and often more than 40 MB for several ScanNet sequences. This means that a 16 MB LLC is simply unable to provide the amount of inter-frame reuse that is required to operate in the 10 mW domain with 5 mm voxels. A notable exception is `table3`, which has a small working set size even with 5 mm voxels. The figure also showcases the diversity from sequence to sequence, with the LLC hit rate ranging from 67% to 98% across the various sequences with 5 mm voxels.

The effects of LLC hit rate are directly observed in average energy per frame, shown in Figure 7.2b. Average energy decreases appreciably at each of the aforementioned inflection points, ultimately stabilizing close to 1 mJ per frame for all sequences with 10 mm voxels. As with LLC hit rate, there is a wide spread in energy per frame with 5 mm voxels. Note that even with a perfect cache, energy consumption is higher with 5 mm voxels due to the extra work.



(a) Time Per Frame

(b) Energy Per Frame

Figure 7.3: Average execution time and energy per frame breakdown for five sequences with an LLC flush at the end of each frame. The left bar in each cluster shows the breakdown with 10 mm voxels, and the right bar shows the breakdown with 5 mm voxels. Bilateral Filter, Visibility Check, Depth Fusion Brick, and Depth Fusion Voxel represent the compute within each unit.

7.4.4 LLC Contention

In practice, our proposed scene reconstruction subsystem will not have exclusive access to the LLC as other components of the XR system will be simultaneously accessing the LLC as well. Specifically, accesses to the LLC by other components may evict scene reconstruction data, reducing hit rate and therefore increasing energy per frame. To model the worst-case scenario of this contention, we repeat our previous experiments and flush the LLC at the end of each frame. Given that we do not expect inter-frame reuse in this scenario, we assume that reads and writes to voxel data bypass the LLC.

Figure 7.3 shows the time and energy breakdown per frame with this new configuration. Even though the hit rate of the LLC decreases considerably (not shown), there is sufficient parallelism in the compute units to hide the latency of DRAM accesses, which results in only a minor slowdown in execution time. In terms of energy, however, the 10 mm configurations observe a 30% increase in energy per frame. Nonetheless, the absolutely energy per frame is still less than 2 mJ, which results in an average power of less than 10 mW. Interestingly, with 5 mm voxels, the energy per frame of the three ScanNet sequences actually *decreases*, as depth image data, which has significant reuse, is no longer repeatedly evicted by voxel data, which has no reuse. Overall, the impact of flushing the LLC is small on both time and energy, as exploiting intra-frame reuse has a larger impact on the system than exploiting inter-frame reuse, as discussed in Section 7.4.3.

7.4.5 Voxel Compression

Given the extreme resource constraints of XR, we believe using 10 mm voxels is an acceptable choice, as it enables ultra-low power operation while still providing reasonable quality meshes [161]. In the future, however, applications and/or users may demand smaller voxel sizes, which as we have just shown, will not execute efficiently with realistic LLC sizes. We conjecture that voxel compression will be able to adequately solve this problem. As an initial experiment, we compressed voxel bricks using `gzip` [204], and obtained a 4× compression ratio. While formats like ASTC [205] can also be used, and in fact are already supported by virtually all modern GPUs, they are optimized for compress-once decompress-many use cases, and as such are not amenable to efficient hardware implementations of compression. Consequently, a new compression scheme would be required for scene reconstruction, one that is able to balance high compression ratios with low-cost hardware for *both* compression and decompression.

7.5 SUMMARY

In this chapter, we showcase an effective and scalable hardware mapping for the various tasks of scene reconstruction. We map bilateral filtering to an ISP, visibility check to a DSP, and voxel fusion to a custom accelerator, striking a balance between reusing existing hardware components and instantiating new ones. Our proposed mapping is able to reduce power consumption to less than 8 mW when used with 10 mm voxels (125× reduction compared to an embedded GPU), and less than 26 mW when used with 5 mm voxels (38× reduction compared to an embedded GPU). Although the power consumption with 5 mm voxels is considerably lower than with an embedded GPU, it can be further optimized if the smaller voxel size becomes necessary due to QoE constraints. To that end, voxel compression is a promising approach, and future work can address the problem of designing a compression scheme that achieves a balance between compression and decompression efficiency and hardware friendliness.

Instead of requiring manual designer effort, in the future, tools can be developed which automatically go through the scalable hardware mapping process described in this chapter. We take the first step in that direction by designing Trireme [21], a design space exploration tool that automatically determines acceleration candidates from a given component. We summarize Trireme in Section 9.5.

CHAPTER 8: RELATED WORK

8.1 SYSTEMS

Project Esky [206] is a partially open-source XR runtime for Project North Star [131]. It uses closed head tracking software from Intel T26x cameras [207] and its communication and runtime framework has limited modularity – it supports a pre-defined set of plugins and does not expose interfaces to enable easy plug-n-play for new components and implementations. It also does not support OpenXR, the growing open XR application standard. Chen et al. [208] characterize AR applications on a commodity smartphone, but neither analyze individual components nor consider futuristic components due to the closed-source nature of the platform. Yi et al. [209] perform scheduling of parallel inference and rendering tasks in smartphone-based AR applications, but do not consider other XR system components due to their closed-source nature.

8.2 BENCHMARKS

There have been other works that have developed benchmark suites for XR. ARBench [210] proposes a set of six AR benchmarks to evaluate mobile devices. However, these benchmarks exist at the application level, provide no visibility into the XR runtime, and all but one focus on rendering. Raytrace in SPLASH-2 [31], VRMark [211], FCAT [212], and Unigine [213] are examples of graphics rendering benchmarks, but do not contain the perception pipeline nor adaptive display components. The SLAMBench [214, 215, 216] series of benchmarks aid the characterization and analysis of available SLAM algorithms, but contain neither the visual pipeline nor the audio pipeline. Unlike these benchmarks, the work presented in this thesis spans the entire system. ARBench can be easily integrated with ILLIXR to provide both custom XR applications and a custom XR runtime.

8.3 SPECIALIZATION

There is an increasing interest in XR in the architecture community and several works have shown promising specialization results for individual XR components: Processing-in-Memory architectures for graphics [217, 218], video accelerators for VR [145, 219, 220, 221, 222], 3D point cloud acceleration [223, 224], stereo acceleration [225, 226], computer vision accelerators [227, 228], specialized sensors [229], scene reconstruction hardware [192, 193],

and SLAM chips [168, 169, 170, 185]. However, these works have been for single components in isolation or for a subset of the entire XR system.

Significant work has also been done on remote rendering [117, 230, 231, 232, 233, 234, 235] and 360 video streaming [236, 237, 238, 239]. While these works do consider full end-to-end systems, they only focus on the rendering aspect of the system and do not consider the interplay between the various components of the system. This thesis instead provides a full-system-benchmark, consisting of a set of XR components representing a full XR workflow, along with insights into their characteristics. Our goal is to propel architecture and systems research in the direction of full domain-specific *system* design.

8.4 ON-SENSOR COMPUTING

Several works have developed entire VIO chips to meet the stringent power and performance requirements of XR and micro aerial vehicles. The most closely related literature to our work is Navion [240], the work by Mandal et al. [170], and the work by Li et al. [169]. Although all three works build VIO chips that achieved beyond-real-time performance while consuming only milliwatts of power, they do not consider I/O power between the chip and sensors. Moreover, while we do not evaluate the security and privacy of our designs in this paper, we believe our designs present an avenue for exploring this aspect in the future.

Furthermore, no previous work has considered on-sensor computing for VIO. Sony recently released a camera that performs image processing on the sensor itself [175]. Feng et al. [14] offload a part of eye tracking to on-sensor compute. Gomez et al. [15] divide a hand tracking neural network into two parts, and execute the more memory intensive one on the sensor. Pinkham et al. [241] adopt a similar approach and also study the execution of DNNs on near-sensor compute. Interestingly, Kodukula et al. [178] show that running intensive DNNs on camera sensors leads to thermal noise which can significantly degrade image quality. They also propose mitigation techniques.

However, performing fully distributed VIO is significantly more complex than processing individual images, as is the case in each of the aforementioned works, and all these existing designs consist of cameras which work in isolation and do not communicate with each other. Furthermore, since our design does not include DRAM, we expect its thermal noise to be low, and thus not affect image quality.

In another work, Pinkham et al. [177] propose a neural compression technique to reduce I/O power. While it can reduce I/O power like our work, it does not alleviate the privacy concerns associated with transmitting camera images to a device with a programmable microcontroller.

8.5 SCENE RECONSTRUCTION

KinectFusion [81] was the first work that supported real-time scene reconstruction using commodity depth sensors. Since KinectFusion, a plethora of works have explored RGB-D-based scene reconstruction over the past decade. Notable among these are methods that improve the scalability of scene reconstruction by using efficient data structures such as octrees and hash tables [179, 180]. These methods work with both ICP-based (Iterative Closest Point) [80, 184] and VI-SLAM-based [84, 85, 183] pose trackers. However, all of these methods run at a constant 30 Hz in all cases, and do not consider dynamically changing fusion frequency based on the scene.

Recently, several papers have investigated using neural approaches for scene reconstruction. Sucar et al. [242] introduce using implicit neural rendering for real-time scene reconstruction using a multilayer perceptron (MLP) for scene representation. Zhu et al. [243] follow their work by proposing hierarchical grid-based neural implicit encoding to better suit larger scenes. While these online approaches provide reasonable reconstructions and predict unobserved regions, they are still experimental. Both methods are compute-heavy and require desktop-grade GPUs to execute in real-time, rendering them unsuitable for resource-constraint mobile systems. Azinovic et al. [244] propose using a neural radiance field (NeRF) for offline scene reconstruction. While their approach can generate high-quality room-scale reconstructions, it consumes an impractical amount of time (9 hours on an NVIDIA RTX 3090 GPU to finish reconstruction), and requires all camera images and poses prior to running.

Accelerators have also been proposed for scene reconstruction. Gautier et al. [192] accelerate InfiniTAM on an FPGA. Since pose estimation is performed by VIO in an XR system, neither ICP nor raycasting are required in scene reconstruction, alleviating the need to design accelerators for them. VoxelCache [193] proposes a custom hardware cache to optimize hash table accesses. However, hash table accesses constitute only a small percentage of the total work performed by the algorithm. Additionally, it is not feasible to modify the L1 caches of the general purpose CPU cores to accelerate one small sub-task of one component, as done in VoxelCache. Instead, AdaptiveFusion significantly reduces the amount of work performed by scene reconstruction, and our hardware mapping reuses existing hardware blocks before designing new ones.

PID controllers have been applied to SLAM in previous work in the context of approximate computing [187, 188]. These controllers observe pose distance or camera velocity to calculate the approximation level for a given frame. Although they do not consider scene reconstruction frequency as an approximation knob, it is a natural extension of their work,

and indeed AdaptiveFusion is inspired by these methods. One notable difference between our work and prior methods is the choice of process variable.

CHAPTER 9: CONCLUSION AND FUTURE WORK

Extended reality is a rich domain that has the potential to transform most aspects of human life. However, it has historically not been possible to conduct end-to-end systems research in XR, and thus derive system design principles, due to a lack of open-source benchmarks and systems. To address this issue, this thesis develops a unique end-to-end open-source XR testbed, and demonstrates the efficacy of several broadly applicable design principles in optimizing key components of the XR system.

First, we present ILLIXR, the first fully open-source full-system Extended Reality testbed containing state-of-the-art components integrated with a modular and extensible runtime framework. ILLIXR provides an OpenXR-compliant interface to applications such as game engines, and reports quality of experience (QoE) metrics, enabling end-to-end characterization and analysis of the full XR system.

By performing both a system-level and a component-level analysis of ILLIXR, the first such analysis to be publicly available, we show several key implications for architects and system designers, including demanding performance, power, and QoE requirements, a large diversity of critical tasks, significant computation variability that challenges scheduling and specialization, and a diversity in bottlenecks throughout the system, from I/O and compute requirements to power and resource contention.

We then show that optimizing the XR system will require applying a diverse set of broadly applicable design principles, where each principle may be suitable for a family of related components. The first principle that we showcase is distributed on-sensor computing, which we use to develop a new hardware architecture for VIO. We perform distributed VIO on the image sensors themselves, and design a caching protocol to minimize data movement between the sensors, which reduces both link power and potentially improves the security and privacy of the design. Our distributed design reduces link power by 37×, VIO chip power by 3.3×, and VIO subsystem power by 1.3× compared to a centralized design.

The second design principle that we demonstrate is online frequency control, which we apply to scene reconstruction to reduce its power consumption. We develop a frequency controller called AdaptiveFusion that dynamically changes the frequency of scene reconstruction based on user motion, reducing power by 4.9× compared to a constant frequency baseline.

The third and final design principle that we employ is scalable hardware mapping, also in the context of scene reconstruction. We carefully map the tasks of the algorithm to an ISP, a DSP, and a custom accelerator, reusing two existing hardware components, and only

requiring one new one. Combined with AdaptiveFusion, our proposed system reduces power by 128×–270× compared to a state-of-the-art baseline running on an embedded GPU.

In summary, this thesis provides the research community with a novel, one-of-a-kind infrastructure and foundational quantitative analyses to enable a new era of research in domain-specific *systems* in general and XR systems in particular. In addition to the infrastructure, this thesis shows the applicability of a variety of general design principles—on-sensor computing, online frequency control, and scalable hardware mapping—to various parts of the XR system.

We expect this thesis to shape a broad and long-term research agenda of the science of designing end-to-end quality driven and end-to-end hardware-software-application co-designed domain-specific systems. ILLIXR and the research it can enable have the potential to particularly transform XR, an emerging domain of critical importance and likely to transform all endeavors of human activity. We conclude by summarizing several research projects that we have started in collaboration with others after the development of ILLIXR, as well as other research projects enabled by this thesis, including those that apply the design principles presented in this thesis to the broader XR system.

9.1 COMPUTE OFFLOAD

At the moment, all ILLIXR components run on the edge device. However, it is neither feasible to run all components on the edge device due to the limited power budget nor required since some components are inherently latency-tolerant. Thus, offloading compute to an edge or cloud server is a design principle that is applicable to several components within an XR system.

In initial work [17], we show that the IMU integrator is able to compensate for surprisingly long head tracking latencies, paving the way for offloading head tracking to a server. We design a system in which the client device communicates over a 5G network with a server running VIO. While maintaining an acceptable QoE, offloading VIO reduces CPU power by 2.5×, total power by 1.3×, and, unlike hardware accelerators, retains the ability to change the algorithm.

In the future, other components can be offloaded as well; e.g., AdaptiveFusion reduces the frequency of scene reconstruction sufficiently to enable it to be offloaded to the cloud. Offloading rendering is already a popular research topic [117, 230, 231, 232, 233, 234, 235]. The end goal of this line of research is to develop a methodology for offloading-driven hardware-software-algorithm codesign, and perform real-time offloading decisions based on device resource usage and network capabilities.

9.2 NETWORK OPTIMIZATION

An immediate challenge that arises with compute offload is management of network traffic, especially when multiple components with different network characteristics are being offloaded. Researchers at Intel are studying the network behavior of various components using ILLIXR, and tackling the problem of network traffic management by using Wireless Time Sensitive Networking (WTSN) [245], a networking technology that allows creating a schedule for different network streams, which can be used to satisfy QoE constraints of the system.

9.3 QOE-DRIVEN AUTOMATED CROSS-COMPONENT APPROXIMATE COMPUTING

Current approximation techniques typically look at component-level or subsystem metrics, whereas it is the end-to-end QoE that ultimately matters in many emerging domains. Determining whether and how a certain approximation will impact the end-to-end QoE, how errors will compose across components, and how to trade off accuracy among components are all unanswered questions, as discussed in Section 3.5. Given ILLIXR's end-to-end nature, we can begin answering these questions.

In an end-to-end system, it is possible to approximate one component and compensate for some of the resultant errors in another. By carefully choosing both the approximation and its remedy, power consumption can be decreased. We term this design principle QoE-aware cross-component approximate computing. In the first exploration of this technique [246], we study the power, performance, and quality tradeoffs involved in implementing eye-tracked foveated rendering.

We optimize the eye tracker via principled pruning of its neural network, and compensate for the accuracy loss by choosing a larger fovea in the foveated renderer. The final pruned eye tracker has a degree tracking error of several degrees, but a larger fovea ensures that the user's gaze does not fall onto blurry parts of the scene. Overall, while incurring negligible quality loss, our pruned eye tracker is 20× more energy-efficient than the baseline eye tracker, and our foveated renderer is 1.7× more energy-efficient compared to a fixed foveated renderer (which keeps the fovea in the middle of the display regardless of where the user is looking). Our eventual goal is to develop disciplined end-to-end QoE-driven approximation techniques and methodologies similar to ApproxTuner [247], and apply them to the entire XR system in a principled manner.

9.4 QOE-DRIVEN SCHEDULING

As discussed in Section 3.5, in QoE-driven emerging domains, tasks have to be scheduled and resources managed to meet one or more QoE metrics. ILLIXR is well-suited for studying this phenomenon as ILLIXR's task graph is a DAG with multiple critical paths and QoE constraints. To that end, a natural next step is to apply online frequency control from Chapter 6 to other components of the XR system, and develop a framework that automatically determines an optimal frame rate of each component, and schedules components to meet QoE for a given hardware mapping.

In the first milestone of this approach, we develop Catan [19, 20], a scheduling framework that calculates frame rates of different components of an application, and dynamically allocates CPUs (spatial scheduling) and CPU time (temporal scheduling) to them in order to meet a given QoE requirement. Catan is able to maintain a motion-to-photon latency of less than 20 ms even when running a subset of ILLIXR on just one CPU core. In the future, Catan can be extended to support other hardware components, such as GPUs and accelerators, and to take multiple QoE metrics into account.

9.5 AUTOMATED SELECTION & GENERATION OF ACCELERATORS

As discussed in Section 4.4, it is infeasible to design a unique accelerator for every task within each component. Coupled with the growing number and complexity of tasks and components in the XR system, it is vital to develop techniques which can automatically determine common compute patterns across components to enable hardware reuse, and also generate accelerator software and hardware when new hardware is necessary. Such tools would essentially perform the scalable hardware mapping described in Chapter 7 in an automated fashion, and would apply it to all components of the system.

In our initial work, the Trireme tool uses the HPVM IR representation [23] to guide a design space exploration that considers the exposed loop, task, and streaming parallelism to select accelerators for a subset of ILLIXR components [21]. Being aware of these types of parallelism enables Trireme to generate hardware that is up to 6× faster than that generated by AccelSeeker [22], the previous state-of-the-art tool.

Moving forward, Trireme can be extended to consider multiple components simultaneously, to find a balance between reusing existing hardware blocks and instantiating new ones, and to generate the software required to orchestrate execution among multiple hardware blocks.

9.6 REPRESENTING HETEROGENEOUS PARALLELISM IN SOFTWARE

The compiler intermediate representation (IR) is critical for performance and portability. Traditional IRs such as LLVM do not capture the parallelism or heterogeneity that is prevalent in current hardware architectures. HPVM [23] is a compiler IR that uses a hierarchical dataflow graph (with side effects) to capture several types of parallelism: task, streaming, nested, data, and fine-grained vector parallelism. The hierarchical dataflow graph nodes naturally and flexibly map to potentially heterogeneous compute elements and the edges represent communication between the elements. There is an ongoing effort to compile all of ILLIXR to HPVM, providing a rich representation to develop techniques that can be applied systematically to the entire system, including techniques for a compiler and runtime to perform automated accelerator selection as described in Section 9.5, software and hardware approximations, and local and distributed resource mapping for XR and other similar domain-specific systems.

9.7 ACCELERATOR COMMUNICATION INTERFACES

Future SoCs will require multiple heterogeneous accelerators running in parallel to meet QoE of domains like XR. A shared memory programming model would be able to alleviate the need for programming complex DMA engines to explicitly orchestrate data movement between these accelerators. However, as discussed in Section 4.4, how to design cache coherence protocols, memory consistency models, and the memory and communication fabric of the SoC are open questions regarding the implementation of shared memory in a heterogeneous environment. To answer these questions, we are building upon the Spandex [248] heterogeneous coherence interface for coherence specialization, and using ILLIXR's diverse range of communication patterns to drive the design.

9.8 SPECIALIZED MEMORY HIERARCHIES

In addition to coherence specialization, we are also researching novel on-chip memory hierarchies using the audio playback component of ILLIXR. There are several tasks in audio playback, with varying communication patterns between each one, making it a complex yet standalone driver for such research. Our goal is to develop a methodology for choosing and instantiating memory primitives such as Stash [249], Buffets [250], and Accelerator Store [251] in order to optimize on-chip data movement between accelerators.

9.9 MULTIPARTY XR

Currently ILLIXR supports a single end-user device. The full potential of XR is in multi-user applications such as telepresence. ILLIXR is being expanded to support networked multiparty applications by integrating it with ARENA [252], a distributed XR system that enables multiple users to have shared XR experiences. The end result will be a full stack multiparty XR system that will be fully open-source, enabling study of distributed XR applications and allowing for cross-stack optimizations.

9.10 OTHER RESEARCH DIRECTIONS

There are several other projects using or considering ILLIXR, including for AR security and privacy, low latency networks for XR, use of integrated sensing and compute through 2.5D and 3D packaging techniques, QoE metrics for XR, cross-component codesign driving novel XR algorithms, and simulation techniques to use ILLIXR to drive novel architectures.

9.11 XR SYSTEM DESIGN METHODOLOGY

We have thus far enumerated several design principles that are each applicable to large parts of the end-to-end XR system; namely, on-sensor computing, online frequency control, scalable hardware mapping, compute offload, and cross-component approximate computing. We have applied these principles to various components manually, showing their efficacy. However, in order to tackle the full complexity of the XR system, a systematic methodology is required to apply these principles in a generalizable way. This methodology would include both determining which principles are applicable to which component (e.g., which components can benefit from on-sensor computing), and how the principles should be composed together to optimize the component in the context of the entire system (e.g., what is the interplay between moving compute to the sensor and reusing existing hardware). The end goal would be a principled framework which architects can use to design end-to-end XR systems, and we believe ILLIXR and the insights and approaches described in this thesis provide a solid foundation to achieve that goal.

REFERENCES

[1] S. Adve, R. Bodik, and L. Ceze, "I-USHER: Interfaces to Unlock the Specialized HardwarE Revolution," DARPA Information Science and Technology (ISAT), Tech. Rep., Apr. 2019. [Online]. Available: https://rsim.cs.illinois.edu/Talks/I-USHER.pdf

[2] S. Stoller, M. Carbin, S. V. Adve, K. Agrawal, G. Blelloch, D. Stanzione, K. Yelick, and M. Zaharia, "Future Directions for Parallel and Distributed Computing," in *Report of an NSF workshop to influence the successor to the Scalable Parallelism in the Extreme (SPX) program*, 2019, Report of an NSF workshop to influence the successor to the Scalable Parallelism in the Extreme (SPX) program. [Online]. Available: rsim.cs.illinois.edu/Pubs/SPX_2019_Workshop_Report.pdf

[3] Morgan McGuire, "Exclusive: How NVIDIA Research is Reinventing the Display Pipeline for the Future of VR, Part 1," 2017. [Online]. Available: https://www.roadtovr.com/exclusive-how-nvidia-research-is-reinventing-the-display-pipeline-for-the-future-of-vr-part-1/

[4] R. Aggarwal, T. P. Grantcharov, J. R. Eriksen, D. Blirup, V. B. Kristiansen, P. Funch-Jensen, and A. Darzi, "An evidence-based virtual reality training program for novice laparoscopic surgeons," *Annals of surgery*, vol. 244, no. 2, pp. 310–314, Aug 2006. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/16858196

[5] S. Krohn, J. Tromp, E. M. Quinque, J. Belger, F. Klotzsche, S. Rekers, P. Chojecki, J. de Mooij, M. Akbal, C. McCall, A. Villringer, M. Gaebler, C. Finke, and A. Thöne-Otto, "Multidimensional Evaluation of Virtual Reality Paradigms in Clinical Neuropsychology: Application of the VR-Check Framework," *J Med Internet Res*, vol. 22, no. 4, p. e16724, Apr 2020. [Online]. Available: https://doi.org/10.2196/16724

[6] D. Van Krevelen and R. Poelman, "A survey of augmented reality technologies, applications and limitations," *International journal of virtual reality*, vol. 9, no. 2, pp. 1–20, 2010.

[7] The Khronos Group Inc., "The OpenXR Specification," Available at https://www.khronos.org/registry/OpenXR/specs/1.0/html/xrspec.html (accessed April 5, 2020), Mar. 2020, version 1.0.8.

[8] "Monado - Open Source XR Platform," Available at https://monado.dev/ (accessed April 5, 2020). [Online]. Available: https://monado.dev/

[9] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, B. Tian, H. Yuan, J. Zhang, and S. V. Adve, "ILLIXR: Enabling End-to-End Extended Reality Research," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 24–38.

[10] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, B. Tian, H. Yuan, J. Zhang, and S. V. Adve, "ILLIXR: An Open Testbed to Enable Extended Reality Systems Research," *IEEE Micro*, vol. 42, no. 4, pp. 97–106, 2022.

[11] P. Hübner, K. Clintworth, Q. Liu, M. Weinmann, and S. Wursthorn, "Evaluation of HoloLens Tracking and Depth Sensing for Indoor Mapping Applications," *Sensors (Basel, Switzerland)*, vol. 20, 2020.

[12] M. Huzaifa and M. Kal, "A Distributed Co-designed Sensor-Compute Architecture for Energy-Efficient Visual-Inertial Odometry," in *Submission*, 2022.

[13] M. Kal, A. Alaghi, V. Lee, R. A. Newcombe, A. Suleiman, and M. Huzaifa, "Distributed sensor module for tracking," U.S. Patent 11 182 647, Nov. 23, 2021. [Online]. Available: https://patents.google.com/patent/US11182647B2/en

[14] Y. Feng, N. Goulding-Hotta, A. Khan, H. Reyserhove, and Y. Zhu, "Real-Time Gaze Tracking with Event-Driven Eye Segmentation," in *2022 IEEE on Conference Virtual Reality and 3D User Interfaces (VR)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2022. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/VR51125.2022.00059 pp. 399–408.

[15] J. Gomez, S. Patel, S. S. Sarwar, Z. Li, R. Capoccia, Z. Wang, R. Pinkham, A. Berkovich, T.-H. Tsai, B. de Salvo, and C. Liu, "Distributed On-Sensor Compute System for AR/VR Devices: A Semi-Analytical Simulation Framework for Power Estimation," *ArXiv*, vol. abs/2203.07474, 2022.

[16] M. Huzaifa, B. Tian, Y. Pang, H. Che, S. Wang, and S. Adve, "AdaptiveFusion: Low Power Scene Reconstruction," in *Submission*, 2022.

[17] Q. Jiang, M. Huzaifa, W. Sentosa, J. Zhang, S. Gao, Y. Pang, B. Godfrey, H. Hassanieh, and S. Adve, "Offloading Visual-Inertial Odometry for Low Power Extended Reality," in *Submission*, 2022.

[18] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder, "Foveated 3D Graphics," *ACM Trans. Graph.*, vol. 31, no. 6, nov 2012. [Online]. Available: https://doi.org/10.1145/2366145.2366183

[19] A. Partap, S. Grayson, M. Huzaifa, S. Adve, B. Godfrey, S. Gupta, K. Hauser, and R. Mittal, "On-Device CPU Scheduling for Sense-React Systems," 2022. [Online]. Available: https://arxiv.org/abs/2207.13280

[20] A. Partap, S. Grayson, M. Huzaifa, S. Adve, B. Godfrey, S. Gupta, K. Hauser, and R. Mittal, "On-Device CPU Scheduling for Robot Systems," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022.

[21] G. Zacharopoulos et al., "Trireme: Exploring Hierarchical Multi-Level Parallelism for Domain Specific Hardware Acceleration," 2022.

[22] G. Zacharopoulos, L. Ferretti, G. Ansaloni, G. Di Guglielmo, L. Carloni, and L. Pozzi, "Compiler-Assisted Selection of Hardware Acceleration Candidates from Application Source Code," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, 2019, pp. 129–137.

[23] M. Kotsifakou, P. Srivastava, M. D. Sinclair, R. Komuravelli, V. Adve, and S. Adve, "HPVM: Heterogeneous Parallel Virtual Machine," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3178487.3178493 p. 68–80.

[24] S. Zhao, H. Zhang, C. S. Mishra, S. Bhuyan, Z. Ying, M. T. Kandemir, A. Sivasubramaniam, and C. Das, "HoloAR: On-the-Fly Optimization of 3D Holographic Processing for Augmented Reality," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3466752.3480056 p. 494–506.

[25] N. S. Foundation, "CCRI: New: An Open End-to-End Extended Reality System Infrastructure: Enabling Domain-Specific Edge Systems Research." [Online]. Available: https://www.nsf.gov/awardsearch/showAward?AWD_ID=2120464&HistoricalAwards=false

[26] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: Association for Computing Machinery, 2008. [Online]. Available: https://doi.org/10.1145/1454115.1454128 p. 72–81.

[27] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[28] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, p. 1–17, Sep. 2006. [Online]. Available: https://doi.org/10.1145/1186736.1186737

[29] J. Bucek, K.-D. Lange, and J. v. Kistowski, "SPEC CPU2017: Next-Generation Compute Benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3185768.3185771 p. 41–42.

[30] J. P. Singh, W.-D. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory," *SIGARCH Comput. Archit. News*, vol. 20, no. 1, p. 5–44, Mar. 1992. [Online]. Available: https://doi.org/10.1145/130823.130824

[31] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, ser. ISCA '95. New York, NY, USA: Association for Computing Machinery, 1995. [Online]. Available: https://doi.org/10.1145/223982.223990 p. 24–36.

[32] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros, "Splash-3: A properly synchronized benchmark suite for contemporary research," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 101–111.

[33] S. V. Adve and M. Huzaifa, "An Open-Source Testbed to Democratize Extended Reality Research, Development, and Benchmarking," NVIDIA GPU Technology Conference (GTC), April 2021. [Online]. Available: https://www.youtube.com/watch?v=ZY98lWksnpM

[34] "ILLIXR Consortium," https://illixr.org, 2021.

[35] Qualcomm, "Snapdragon 835 Mobile Platform," https://www.qualcomm.com/products/snapdragon-835-mobile-platform.

[36] Qualcomm, "APQ8009 Processor," https://www.qualcomm.com/products/apq8009.

[37] D. Takahashi, "Oculus chief scientist Mike Abrash still sees the rosy future through AR/VR glasses," https://venturebeat.com/2018/09/26/oculus-chief-scientist-mike-abrash-still-sees-the-rosy-future-through-ar-vr-glasses/, September 2018.

[38] P. Warden, "Smartphone Energy Consumption," https://petewarden.com/2015/10/08/smartphone-energy-consumption/, October 2015.

[39] A. Carroll, "Understanding and Reducing Smartphone Energy Consumption," https://ts.data61.csiro.au/publications/papers/Carroll:phd.pdf, May 2017.

[40] Varjo, "Varjo VR-3," 2020. [Online]. Available: https://varjo.com/products/vr-3/

[41] D. Kanter, "Graphics Processing Requirements for Enabling Immersive VR," *Whitepaper*, 2015.

[42] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward Low-Latency and Ultra-Reliable Virtual Reality," *IEEE Network*, vol. 32, no. 2, pp. 78–84, 2018.

[43] Microsoft, "Microsoft HoloLens 2," https://www.microsoft.com/en-us/hololens/hardware, 2019.

[44] Wikipedia contributors, "HoloLens 2 — Wikipedia, The Free Encyclopedia," https://en.wikipedia.org/wiki/HoloLens_2, 2019.

[45] L. Goode, "The HoloLens 2 Puts a Full-Fledged Computer on Your Face," https://www.wired.com/story/microsoft-hololens-2-headset/, Feb 2019.

[46] Skarredghost, "All you need to know on HoloLens 2," https://skarredghost.com/2019/02/24/all-you-need-know-hololens-2/, May 2019.

[47] Rebecca Pool, "AR/VR/MR 2020: The Future Now Arriving," http://microvision.blogspot.com/2020/02/arvrmr-2020-future-now-arriving.html, 2017. [Online]. Available: http://microvision.blogspot.com/2020/02/arvrmr-2020-future-now-arriving.html

[48] Ian Cutress, "Hot Chips 31 Live Blogs: Microsoft Hololens 2.0 Silicon," https://www.anandtech.com/show/14775/hot-chips-31-live-blogs-microsoft-hololens-20-silicon, 2019.

[49] R. Smith and A. Frumusanu, "The Snapdragon 845 Performance Preview: Setting the Stage for Flagship Android 2018," https://www.anandtech.com/show/12420/snapdragon-845-performance-preview/4, 2018.

[50] K. Hinum, "Qualcomm Snapdragon 855 SoC - Benchmarks and Specs," 2019. [Online]. Available: https://www.notebookcheck.net/Qualcomm-Snapdragon-855-SoC-Benchmarks-and-Specs.375436.0.html

[51] I. Cutress, "Spotted: Qualcomm Snapdragon 8cx Wafer on 7nm," https://www.anandtech.com/show/13687/qualcomm-snapdragon-8cx-wafer-on-7nm, 2018.

[52] K. Matsuhashi, T. Kanamoto, and A. Kurokawa, "Thermal Model and Countermeasures for Future Smart Glasses," *Sensors*, vol. 20, no. 5, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/5/1446

[53] Morgan McGuire, "Exclusive: How NVIDIA Research is Reinventing the Display Pipeline for the Future of VR, Part 2," 2017. [Online]. Available: https://www.roadtovr.com/exclusive-nvidia-research-reinventing-display-pipeline-future-vr-part-2/

[54] Stereolabs, "ZED Software Development Kit," 2020. [Online]. Available: https://www.stereolabs.com/developers/release/

[55] Luxonis, "DepthAI API Documentation," 2022. [Online]. Available: https://docs.luxonis.com/projects/api/en/latest/

[56] Intel, "IntelRealSense," https://github.com/IntelRealSense/librealsense, 2015.

[57] "OpenVINS Repository," https://github.com/rpng/open_vins, 2019.

[58] "ORB-SLAM3 Repository," https://github.com/UZ-SLAMLab/ORB_SLAM3, 2021.

[59] "KimeraVIO Repository," https://github.com/MIT-SPARK/Kimera-VIO, 2017.

[60] "GTSAM Repository," https://github.com/borglab/gtsam, 2020. [Online]. Available: https://github.com/borglab/gtsam

[61] A. K. Chaudhary, R. Kothari, M. Acharya, S. Dangi, N. Nair, R. Bailey, C. Kanan, G. Diaz, and J. B. Pelz, "RITnet: Real-time Semantic Segmentation of the Eye for Gaze Tracking," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 3698–3702.

[62] "InfiniTAM v3 Repository," https://github.com/victorprad/InfiniTAM, 2018.

[63] "KinectFusionApp Repository," https://github.com/chrdiller/KinectFusionApp, 2018.

[64] "ElasticFusion Repository," http://github.com/mp3guy/ElasticFusion/, 2015.

[65] J. van Waveren, "The Asynchronous Time Warp for Virtual Reality on Consumer Hardware," in *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*. ACM, 2016, pp. 37–46.

[66] M. Persson, D. Engström, and M. Goksör, "Real-time generation of fully optimized holograms for optical trapping applications," in *Optical Trapping and Optical Micromanipulation VIII*, K. Dholakia and G. C. Spalding, Eds., vol. 8097, International Society for Optics and Photonics. SPIE, 2011. [Online]. Available: https://doi.org/10.1117/12.893599 pp. 291 – 299.

[67] "libspatialaudio Repository," https://github.com/videolabs/libspatialaudio, 2019.

[68] Stereolabs, "ZED Mini - Mixed-Reality Camera," 2018. [Online]. Available: https://www.stereolabs.com/zed-mini/

[69] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[70] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "OpenVINS: A Research Platform for Visual-Inertial Estimation," *IROS 2019 Workshop on Visual-Inertial Navigation: Challenges and Applications*, 2019.

[71] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020. [Online]. Available: https://github.com/MIT-SPARK/Kimera

[72] C. Campos, R. Elvira, J. J. Gómez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[73] A. I. Mourikis and S. I. Roumeliotis, "A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, April 2007, pp. 3565–3572.

[74] L. Carlone, Z. Kira, C. Beall, V. Indelman, and F. Dellaert, "Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 4290–4297.

[75] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation," *Robotics: Science and Systems (RSS) conference*, 01 2015.

[76] S. J. Garbin, Y. Shen, I. Schuetz, R. Cavin, G. Hughes, and S. S. Talathi, "OpenEDS: Open Eye Dataset," 2019.

[77] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[78] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," 2016.

[79] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb, "State of the art on 3D reconstruction with RGB-D cameras," in *Computer graphics forum*, vol. 37, no. 2. Wiley Online Library, 2018, pp. 625–652.

[80] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. H. S. Torr, and D. W. Murray, "Very High Frame Rate Volumetric Integration of Depth Images on Mobile Device," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 11, 2015.

[81] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[82] T. Whelan, S. Leutenegger, R. Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM Without A Pose Graph," in *Robotics: Science and Systems*, 2015.

[83] P. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[84] L. Han and L. Fang, "FlashFusion: Real-time Globally Consistent Dense 3D Reconstruction using CPU Computing," in *Robotics: Science and Systems XIV*, 06 2018.

[85] K. Wang, F. Gao, and S. Shen, "Real-time Scalable Dense Surfel Mapping," in *2019 International Conference on Robotics and Automation (ICRA)*, 05 2019, pp. 6919–6925.

[86] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration," *ACM Trans. Graph.*, vol. 36, no. 3, may 2017. [Online]. Available: https://doi.org/10.1145/3054739

[87] S. M. LaValle, A. Yershova, M. Katsev, and M. Antonov, "Head tracking for the Oculus Rift," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 187–194.

[88] D. Wagner, "Motion to Photon Latency in Mobile AR and VR," https://medium.com/\@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c480926, 2018. [Online]. Available: https://medium.com/\@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c480926

[89] D. Evangelakos and M. Mara, "Extended TimeWarp latency compensation for virtual reality," in *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2016.

[90] L. McMillan and G. Bishop, "Head-tracked stereoscopic display using image warping," in *Stereoscopic Displays and Virtual Reality Systems II*, S. S. Fisher, J. O. Merritt, and M. T. Bolas, Eds., vol. 2409, International Society for Optics and Photonics. SPIE, 1995. [Online]. Available: https://doi.org/10.1117/12.205865 pp. 21 – 30.

[91] A. Kore, "Display technologies for Augmented and Virtual Reality," 2018. [Online]. Available: https://medium.com/inborn-experience/isplay-technologies-for-augmented-and-virtual-reality-82feca4e909f

[92] "van Waveren Oculus Demo Repository," https://github.com/KhronosGroup/Vulkan-Samples-Deprecated/tree/master/samples/apps/atw, 2017. [Online]. Available: https://github.com/KhronosGroup/Vulkan-Samples-Deprecated/tree/master/samples/apps/atw

[93] F. Sinclair, "Spatio-temporal reprojection for virtual and augmented reality applications," Bachelor's Thesis, University of Illinois at Urbana-Champaign, 2020.

[94] NVIDIA, "Deep Learning Super Sampling," 2022. [Online]. Available: https://www.nvidia.com/en-us/geforce/technologies/dlss/

[95] AMD, "AMD FidelityFX Super Resolution," 2022. [Online]. Available: https://www.amd.com/en/technologies/fidelityfx-super-resolution

[96] G. Kramida, "Resolving the Vergence-Accommodation Conflict in Head-Mounted Displays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 7, pp. 1912–1931, July 2016.

[97] J.-H. R. Chang, B. V. K. V. Kumar, and A. C. Sankaranarayanan, "Towards Multifocal Displays with Dense Focal Stacks," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 198:1–198:13, Dec. 2018. [Online]. Available: http://doi.acm.org/10.1145/3272127.3275015

[98] N. Padmanaban, R. Konrad, E. A. Cooper, and G. Wetzstein, "Optimizing VR for All Users Through Adaptive Focus Displays," in *ACM SIGGRAPH 2017 Talks*, ser. SIGGRAPH '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3084363.3085029 pp. 77:1–77:2.

[99] P. R. K. Turnbull and J. R. Phillips, "Ocular effects of virtual reality headset wear in young adults," in *Scientific Reports*, 2017.

[100] A. Maimone, A. Georgiou, and J. S. Kollin, "Holographic near-eye displays for virtual and augmented reality," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 85, 2017.

[101] S. Liu and H. Hua, "A systematic method for designing depth-fused multi-focal plane three-dimensional displays," *Opt. Express*, vol. 18, no. 11, pp. 11 562–11 573, May 2010. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-18-11-11562

[102] N. Matsuda, A. Fix, and D. Lanman, "Focal Surface Displays," *ACM Trans. Graph.*, vol. 36, no. 4, jul 2017. [Online]. Available: https://doi.org/10.1145/3072959.3073590

[103] G. A. Koulieris, K. Akşit, M. Stengel, R. K. Mantiuk, K. Mania, and C. Richardt, "Near-Eye Display and Tracking Technologies for Virtual and Augmented Reality," *Computer Graphics Forum*, vol. 38, no. 2, pp. 493–519, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13654

[104] R. D. Leonardo, F. Ianni, and G. Ruocco, "Computer generation of optimal holograms for optical trap arrays," *Opt. Express*, vol. 15, no. 4, pp. 1913–1922, Feb 2007. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-15-4-1913

[105] Y. Peng, S. Choi, N. Padmanaban, and G. Wetzstein, "Neural Holography with Camera-in-the-Loop Training," *ACM Trans. Graph.*, vol. 39, no. 6, nov 2020. [Online]. Available: https://doi.org/10.1145/3414685.3417802

[106] L. Shi, B. Li, C. Kim, P. Kellnhofer, and W. Matusik, "Towards real-time photorealistic 3D holography with deep neural networks," *Nature*, vol. 592, Mar. 2021.

[107] K. Kavakli, H. Urey, and K. Akşit, "Learned holographic light transport: invited," *Appl. Opt.*, vol. 61, no. 5, pp. B50–B55, Feb 2022. [Online]. Available: https://opg.optica.org/ao/abstract.cfm?URI=ao-61-5-B50

[108] C. H. Brown and B. J. May, *Comparative Mammalian Sound Localization*. New York, NY: Springer New York, 2005, pp. 124–178. [Online]. Available: https://doi.org/10.1007/0-387-28863-5_5

[109] S. Bertet, J. Daniel, E. Parizet, and O. Warusfel, "Investigation on Localisation Accuracy for First and Higher Order Ambisonics Reproduced Sound Sources," *Acta Acustica united with Acustica*, vol. 99, p. 642 – 657, 07 2013.

[110] A. Vilkaitis, M. Dring, A. Hill, C. Middlicott, and B. Wiggins, "Room Acoustics and Virtual Reality: An implementation of auralisation and 360 degree image techniques to create virtual representations of spaces," in *Reproduced Sound 2016 - Proceedings of the Institute of Acoustics*, 11 2016.

[111] F. Hollerweger, "An Introduction to Higher Order Ambisonic," http://decoy.iki.fi/dsound/ambisonic/motherlode/source/HOA_intro.pdf, 2008.

[112] F. Zotter and M. Frank, "All-Round Ambisonic Panning and Decoding," *J. Audio Eng. Soc*, vol. 60, no. 10, pp. 807–820, 2012. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=16554

[113] M. A. Gerzon, "Practical Periphony: The Reproduction of Full-Sphere Sound," in *Audio Engineering Society Convention 65*, Feb 1980. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=3794

[114] J. Daniel, "Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia," Ph.D. dissertation, Université Paris, 01 2000.

[115] M. Frank, F. Zotter, and A. Sontacchi, "Producing 3D Audio in Ambisonics," in *Audio Engineering Society Conference: 57th International Conference: The Future of Audio Entertainment Technology – Cinema, Television and the Internet*, Mar 2015. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=17605

[116] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[117] J. Meng, S. Paul, and Y. C. Hu, "Coterie: Exploiting Frame Similarity to Enable High-Quality Multiplayer VR on Commodity Mobile Devices," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3373376.3378516 p. 923–937.

[118] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," *IEEE Transactions on Mobile Computing*, 2019.

[119] A. Schollmeyer, S. Schneegans, S. Beck, A. Steed, and B. Froehlich, "Efficient hybrid image warping for high frame-rate stereoscopic rendering," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 4, pp. 1332–1341, 2017.

[120] P. Andersson, T. Akenine-Möller, J. Nilsson, K. Åström, M. Oskarsson, and M. Fairchild, "FLIP: A Difference Evaluator for Alternating Images," *Proceedings of the ACM in Computer Graphics and Interactive Techniques*, vol. 3, no. 2, 2020.

[121] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry. In 2018 IEEE," in *RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 7244–7251.

[122] Z. Li, C. Bampis, J. Novak, A. Aaron, K. Swanson, A. Moorthy, and J. De Cock, "VMAF: The Journey Continues," https://netflixtechblog.com/vmaf-the-journey-continues-44b51ee9ed12, October 2018.

[123] C. G. Bampis and A. C. Bovik, "Learning to Predict Streaming Video QoE: Distortions, Rebuffering and Memory," *ArXiv*, vol. abs/1703.00633, 2017.

[124] R. K. Mantiuk, G. Denes, A. Chapiro, A. Kaplanyan, G. Rufo, R. Bachy, T. Lian, and A. Patney, "FovVideoVDP: A Visible Difference Predictor for Wide Field-of-View Video," *ACM Trans. Graph.*, vol. 40, no. 4, jul 2021. [Online]. Available: https://doi.org/10.1145/3450626.3459831

[125] M. Narbutt, J. Skoglund, A. Allen, M. Chinen, D. Barry, and A. Hines, "AMBIQUAL: Towards a Quality Metric for Headphone Rendered Compressed Ambisonic Spatial Audio," *Applied Sciences*, vol. 10, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/9/3188

[126] NVIDIA, "NVIDIA AGX Xavier," https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit, 2017. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit

[127] M. Leap, "Magic Leap 1," 2019. [Online]. Available: https://www.magicleap.com/en-us/magic-leap-1

[128] NVIDIA, "NVIDIA TX2," 2017. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/

[129] F. Abazovic, "Snapdragon 835 is a 10nm slap in Intel's face ," 2018. [Online]. Available: https://www.fudzilla.com/news/mobile/42154-snapdragon-835-in-10nm-is-slap-in-intel-s-face

[130] Oculus, "Oculus Quest: All-in-One VR Headset," https://www.oculus.com/quest/?locale=en_US, 2019. [Online]. Available: https://www.oculus.com/quest/?locale=en_US

[131] L. Motion, "Project North Star," 2020. [Online]. Available: https://github.com/leapmotion/ProjectNorthStar

[132] Monado, "Sponza scene in Godot with OpenXR addon," https://gitlab.freedesktop.org/monado/demos/godot-sponza-openxr, 2019.

[133] Godot, "Material Testers," https://github.com/godotengine/godot-demo-projects/tree/master/3d/material_testers, 2020.

[134] Godot, "Platformer 3D," https://github.com/godotengine/godot-demo-projects/tree/master/3d/platformer, 2020.

[135] J. Linietsky, A. Manzur, and contributors, "Godot," https://godotengine.org/, 2020.

[136] NVIDIA, "NSight Systems," 2020. [Online]. Available: https://developer.nvidia.com/nsight-systems

[137] Intel, "Intel VTune Profiler," 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/vtune-profiler.html

[138] N/A, "perf," 2022. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[139] NVIDIA, "NSight Compute," 2020. [Online]. Available: https://developer.nvidia.com/nsight-compute

[140] NVIDIA, "NSight Graphics," 2020. [Online]. Available: https://developer.nvidia.com/nsight-graphics

[141] NVIDIA, "NVIDIA System Management Interface," 2020. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[142] G. Ali, S. Bhalachandra, N. Wright, A. Sill, and Y. Chen, "Evaluation of power controls and counters on general-purpose Graphics Processing Units (GPUs)," *Poster: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[143] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Monas: Multi-objective neural architecture search using reinforcement learning," *arXiv preprint arXiv:1806.10332*, 2018.

[144] J. Lew, D. A. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogers et al., "Analyzing machine learning workloads using a detailed GPU simulator," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 151–152.

[145] Y. Leng, C.-C. Chen, Q. Sun, J. Huang, and Y. Zhu, "Energy-efficient Video Processing for Virtual Reality," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: ACM, 2019. [Online]. Available: http://doi.acm.org/10.1145/3307650.3322264 pp. 91–103.

[146] NVIDIA, "Software-Based Power Consumption Modeling." [Online]. Available: https://docs.nvidia.com/jetson/l4t/index.html\#page/Tegra\%20Linux\%20Driver\%20Package\%20Development\%20Guide/power_management_jetson_xavier.html\#wwpID0E0AG0HA

[147] NVIDIA, "Xavier Series Soc Technical Reference Manual." [Online]. Available: https://developer.nvidia.com/embedded/dlc/xavier-series-soc-technical-reference-manual

[148] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[149] B. Bauman and P. Seeling, "Towards Predictions of the Image Quality of Experience for Augmented Reality Scenarios," *arXiv preprint arXiv:1705.01123*, 2017.

[150] Y. Arifin, T. G. Sastria, and E. Barlian, "User experience metric for augmented reality application: a review," *Procedia Computer Science*, vol. 135, pp. 648–656, 2018.

[151] Y. Zhu, X. Min, D. Zhu, K. Gu, J. Zhou, G. Zhai, X. Yang, and W. Zhang, "Toward Better Understanding of Saliency Prediction in Augmented 360 Degree Videos," 2020.

[152] B. Zhang, J. Zhao, S. Yang, Y. Zhang, J. Wang, and Z. Fei, "Subjective and objective quality assessment of panoramic videos in virtual reality environments," in *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2017, pp. 163–168.

[153] H. G. Kim, H.-T. Lim, and Y. M. Ro, "Deep virtual reality image quality assessment with human perception guider for omnidirectional image," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 4, pp. 917–928, 2019.

[154] J. Yang, T. Liu, B. Jiang, H. Song, and W. Lu, "3D panoramic virtual reality video quality assessment based on 3D convolutional neural networks," *IEEE Access*, vol. 6, pp. 38 669–38 682, 2018.

[155] H.-T. Lim, H. G. Kim, and Y. M. Ra, "VR IQA NET: Deep virtual reality image quality assessment using adversarial learning," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 6737–6741.

[156] M. D. Hill and V. J. Reddi, "Accelerator-level Parallelism," *CoRR*, vol. abs/1907.02064, 2019. [Online]. Available: http://arxiv.org/abs/1907.02064

[157] S. Han, B. Liu, R. Cabezas, C. Twigg, P. Zhang, J. Petkau, T.-H. Yu, C.-J. Tai, M. Akbay, Z. Wang, A. Nitzan, G. Dong, Y. Ye, L. Tao, C. Wan, and R. Wang, "MEgATrack: monochrome egocentric articulated hand-tracking for virtual reality," *ACM Transactions on Graphics*, vol. 39, 07 2020.

[158] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib Bin Altaf, N. Vaish, M. Hill, and D. Wood, "The gem5 simulator," *SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, 08 2011.

[159] A. Sembrant, T. E. Carlson, E. Hagersten, and D. Black-Schaffer, "A graphics tracing framework for exploring CPU+GPU memory systems," in *2017 IEEE Intl. Symposium on Workload Characterization (IISWC)*, 2017, pp. 54–65.

[160] T. Field, J. Leibs, J. Bowman, and D. Thomas, "rosbag," 2021. [Online]. Available: http://wiki.ros.org/rosbag

[161] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[162] F. Sinclair, "The VR Museum of Fine Art," 2016. [Online]. Available: https://store.steampowered.com/app/515020/The_VR_Museum_of_Fine_Art/

[163] "Science Teacher Lecturing," May 2013. [Online]. Available: https://freesound.org/people/SpliceSound/sounds/188214/

[164] "Radio Recording," https://freesound.org/people/waveplay./sounds/397000/, July 2017.

[165] "Freesound," https://freesound.org/.

[166] Meta, "Meta Quest Pro Tech Specs," https://www.meta.com/quest/quest-pro/tech-specs/#tech-specs, 2022.

[167] MIPI Alliance, "MIPI Camera Serial Interface 2 (MIPI CSI-2)," https://mipi.org/specifications/csi-2, 2019.

[168] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A 2-mW Fully Integrated Real-Time Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 1106–1119, April 2019.

[169] Z. Li, Y. Chen, L. Gong, L. Liu, D. Sylvester, D. Blaauw, and H. Kim, "An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 134–136.

[170] D. K. Mandal, S. Jandhyala, O. J. Omer, G. S. Kalsi, B. George, G. Neela, S. K. Rethinagiri, S. Subramoney, L. Hacking, J. Radford, E. Jones, B. Kuttanna, and H. Wang, "Visual Inertial Odometry At the Edge: A Hardware-Software Co-design Approach for Ultra-low Latency and Power," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 960–963.

[171] "MIPI I3C & MIPI I3C Basic."

[172] P. Burt and E. Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Transactions on Communications*, vol. 31, no. 4, pp. 532–540, 1983.

[173] Y. Uchida, "Local Feature Detectors, Descriptors, and Image Representations: A Survey," *ArXiv*, vol. abs/1607.08368, 2016.

[174] C. Chang, H. Zhu, M. Li, and S. You, "A Review of Visual-Inertial Simultaneous Localization and Mapping from Filtering-Based and Optimization-Based Perspectives," *Robotics*, vol. 7, p. 45, 08 2018.

[175] I. Spectrum, "Sony Builds AI Into a CMOS Image Sensor," https://spectrum.ieee.org/view-from-the-valley/sensors/imagers/sony-builds-ai-into-a-cmos-image-sensor, 2020.

[176] J. Engel, V. Koltun, and D. Cremers, "Direct Sparse Odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, 2018.

[177] R. Pinkham, T. Schmidt, and A. Berkovich, "Algorithm-Aware Neural Network Based Image Compression for High-Speed Imaging," in *2020 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2020. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/AIVR50618.2020.00040 pp. 196–199.

[178] V. Kodukula, S. Katrawala, B. Jones, C.-J. Wu, and R. LiKamWa, "Dynamic Temperature Management of Near-Sensor Processing for Energy-Efficient High-Fidelity Imaging," *Sensors*, vol. 21, no. 3, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/3/926

[179] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 6, pp. 1–11, 2013.

[180] F. Steinbruecker, J. Sturm, and D. Cremers, "Volumetric 3D Mapping in Real-Time on a CPU," in *Int. Conf. on Robotics and Automation*, Hongkong, China, 2014.

[181] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 343–352.

[182] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "ElasticFusion: Real-time dense SLAM and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[183] A. S. Vempati, I. Gilitschenski, J. Nieto, P. Beardsley, and R. Siegwart, "Onboard real-time dense reconstruction of large-scale environments for UAV," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3479–3486.

[184] T. Whelan, S. Leutenegger, R. Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM Without A Pose Graph," in *Robotics: Science and Systems*, 2015.

[185] Y. Gan, Y. Bo, B. Tian, L. Xu, W. Hu, S. Liu, Q. Liu, Y. Zhang, J. Tang, and Y. Zhu, "Eudoxus: Characterizing and Accelerating Localization in Autonomous Machines Industry Track Paper," in *2021 IEEE Intl. Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 827–840.

[186] K. J. Åström and R. M. Murray, "Feedback systems," in *Feedback Systems*. Princeton university press, 2010.

[187] Y. Pei, S. Biswas, D. S. Fussell, and K. Pingali, "SLAMBooster: An application-aware online controller for approximation in dense SLAM," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2019, pp. 296–310.

[188] Y. Pei, S. Biswas, D. S. Fussell, and K. Pingali, "A Methodology for Principled Approximation in Visual SLAM," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3410463.3414636 p. 373–386.

[189] T. Schöps, T. Sattler, and M. Pollefeys, "BAD SLAM: Bundle Adjusted Direct RGB-D SLAM," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[190] Z. Murez, T. van As, J. Bartolozzi, A. Sinha, V. Badrinarayanan, and A. Rabinovich, "Atlas: End-to-End 3D Scene Reconstruction from Posed Images," in *ECCV*, 2020. [Online]. Available: https://arxiv.org/abs/2003.10432

[191] I. Magaki, M. Khazraee, L. V. Gutierrez, and M. B. Taylor, "ASIC Clouds: Specializing the Datacenter," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 178–190.

[192] Q. Gautier, A. Althoff, and R. Kastner, "FPGA Architectures for Real-time Dense SLAM," in *2019 IEEE 30th Intl. Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160-052X, 2019, pp. 83–90.

[193] S. Durvasula, R. Kiguru, S. Mathur, J. Xu, J. Lin, and N. Vijaykumar, "VoxelCache: Accelerating Online Mapping in Robotics and 3D Reconstruction Tasks," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '22. Association for Computing Machinery, 2022.

[194] M. Saint-Laurent, P. Bassett, K. Lin, B. Mohammad, Y. Wang, X. Chen, M. Alradaideh, T. Wernimont, K. Ayyar, D. Bui, D. Galbi, A. Lester, M. Pedrali-Noy, and W. Anderson, "A 28 nm DSP Powered by an On-Chip LDO for High-Performance and Energy-Efficient Mobile Applications," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 81–91, 2015.

[195] Z. Liu, T. Park, H. Park, and N. Kim, "Ultra-low-power image signal processor for smart camera applications," *Electronics Letters*, vol. 51, no. 22, pp. 1778–1780, 2015. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/el.2015.1924

[196] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996. [Online]. Available: https://doi.org/10.1145/237170.237269 p. 303–312.

[197] L. Codrescu, W. Anderson, S. Venkumanhanti, M. Zeng, E. Plondke, C. Koob, A. Ingle, C. Tabony, and R. Maule, "Hexagon DSP: An Architecture Optimized for Mobile Multimedia and Communications," *Micro, IEEE*, vol. 34, pp. 34–43, 03 2014.

[198] A. Frumusanu, "Apple Announces The Apple Silicon M1: Ditching x86 - What to Expect, Based on A14," 2020. [Online]. Available: https://www.anandtech.com/show/16226/apple-silicon-m1-a14-deep-dive

[199] "MAP - Modeling Architectural Platform," https://github.com/sparcians/map, 2022.

[200] T. Tambe, Private communication, 2022.

[201] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, 2011.

[202] Y. Chen, Y. Fu, M. Lee, S. George, Y. Liu, V. Narayanan, H. Yang, and X. Li, "FAST: A Fully-Concurrent Access Technique to All SRAM Rows for Enhanced Speed and Energy Efficiency in Data-Intensive Applications," 2022. [Online]. Available: https://arxiv.org/abs/2205.11088

[203] D. Wei, T. Anand, G. Shu, J. E. Schutt-Ainé, and P. K. Hanumolu, "A 10-Gb/s/ch, 0.6-pJ/bit/mm Power Scalable Rapid-ON/OFF Transceiver for On-Chip Energy Proportional Interconnects," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 3, pp. 873–883, 2018.

[204] J. loup Gailly and M. Adler, "gzip," 2022. [Online]. Available: https://www.gzip.org/

[205] J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson, "Adaptive Scalable Texture Compression," in *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. The Eurographics Association, 2012.

[206] R. D. Constantine, D. F. Quiros, C. Rodda, B. C. Brown, N. B. Zerkin, and Á. Cassinelli, "Project Esky: Enabling High Fidelity Augmented Reality on an Open Source Platform," *Companion Proceedings of the 2020 Conference on Interactive Surfaces and Spaces*, 2020.

[207] Intel, "Tracking Solutions," 2021. [Online]. Available: https://www.intelrealsense.com/tracking

[208] H. Chen, Y. Dai, H. Meng, Y. Chen, and T. Li, "Understanding the Characteristics of Mobile Augmented Reality Applications," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, April 2019, pp. 128–138.

[209] J. Yi and Y. Lee, "Heimdall: mobile GPU coordination platform for augmented reality applications," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[210] S. Chetoui, R. Shahi, S. Abdelaziz, A. Golas, F. Hijaz, and S. Reda, "ARBench: Augmented Reality Benchmark For Mobile Devices," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022, pp. 242–244.

[211] "VRMark," 2016. [Online]. Available: https://benchmarks.ul.com/vrmark

[212] "NVIDIA Frame Capture Analysis Tool," 2017. [Online]. Available: https://www.geforce.com/hardware/technology/fcat

[213] "Unigine Superposition Benchmark," 2017. [Online]. Available: https://benchmark.unigine.com/superposition

[214] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Luján, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber, "Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5783–5790.

[215] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Lujan, S. Furber, A. J. Davison, P. H. J. Kelly, and M. F. P. O'Boyle, "SLAMBench2: Multi-Objective Head-to-Head Benchmarking for Visual SLAM," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3637–3644.

[216] M. Bujanca, P. Gafton, S. Saeedi, A. Nisbet, B. Bodin, M. F. P. O'Boyle, A. J. Davison, P. H. J. Kelly, G. Riley, B. Lennox, M. Luján, and S. Furber, "SLAMBench 3.0: Systematic Automated Reproducible Evaluation of SLAM Systems for Robot Vision Challenges and Scene Understanding," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6351–6358.

[217] C. Xie, S. L. Song, J. Wang, W. Zhang, and X. Fu, "Processing-in-Memory Enabled Graphics Processors for 3D Rendering," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 637–648.

[218] C. Xie, X. Zhang, A. Li, X. Fu, and S. Song, "PIM-VR: Erasing Motion Anomalies In Highly-Interactive Virtual Reality World with Customized Memory Cube," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2019, pp. 609–622.

[219] Y. Leng, C.-C. Chen, Q. Sun, J. Huang, and Y. Zhu, "Semantic-Aware Virtual Reality Video Streaming," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, ser. APSys '18. New York, NY, USA: ACM, 2018. [Online]. Available: http://doi.acm.org/10.1145/3265723.3265738 pp. 21:1–21:7.

[220] A. Mazumdar, A. Alaghi, J. T. Barron, D. Gallup, L. Ceze, M. Oskin, and S. M. Seitz, "A Hardware-friendly Bilateral Solver for Real-time Virtual Reality Video," in *Proceedings of High Performance Graphics*, ser. HPG '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3105762.3105772 pp. 13:1–13:10.

[221] A. Mazumdar, T. Moreau, S. Kim, M. Cowan, A. Alaghi, L. Ceze, M. Oskin, and V. Sathe, "Exploring computation-communication tradeoffs in camera systems," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2017, pp. 177–186.

[222] P. Ranganathan, D. Stodolsky, J. Calow, J. Dorfman, M. Guevara, C. W. Smullen IV, A. Kuusela, R. Balasubramanian, S. Bhatia, P. Chauhan et al., "Warehouse-scale video acceleration: co-design and deployment in the wild," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 600–615.

[223] T. Xu, B. Tian, and Y. Zhu, "Tigris: Architecture and Algorithms for 3D Perception in Point Clouds," in *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: ACM, 2019. [Online]. Available: http://doi.acm.org/10.1145/3352460.3358259 pp. 629–642.

[224] Y. Feng, B. Tian, T. Xu, P. Whatmough, and Y. Zhu, "Mesorasi: Architecture Support for Point Cloud Analytics via Delayed-Aggregation," *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2020. [Online]. Available: http://dx.doi.org/10.1109/MICRO50266.2020.00087

[225] J. Choi, E. P. Kim, R. A. Rutenbar, and N. R. Shanbhag, "Error resilient MRF message passing architecture for stereo matching," in *SiPS 2013 Proceedings*, 2013, pp. 348–353.

[226] Y. Feng, P. Whatmough, and Y. Zhu, "ASV: Accelerated Stereo Vision System," in *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: ACM, 2019. [Online]. Available: http://doi.acm.org/10.1145/3352460.3358253 pp. 643–656.

[227] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough, "Euphrates: Algorithm-SoC Co-design for Low-power Mobile Continuous Vision," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. Piscataway, NJ, USA: IEEE Press, 2018. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00052 pp. 547–560.

[228] S. Triest, D. Nikolov, J. Rolland, and Y. Zhu, "Co-Optimization of Optics, Architecture, and Vision Algorithms," in *Workshop on Approximate Computing Across the Stack (WAX)*, 2019.

[229] V. Kodukula, A. Shearer, V. Nguyen, S. Lingutla, Y. Liu, and R. LiKamWa, "Rhythmic pixel regions: multi-resolution visual sensing system towards high-precision visual computing at low power," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 573–586.

[230] S. Zhao, H. Zhang, S. Bhuyan, C. S. Mishra, Z. Ying, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "Deja View: Spatio-Temporal Compute Reuse for Energy-Efficient 360 VR Video Streaming," in *Proceedings of the 47th International Symposium on Computer Architecture*, ser. ISCA '20, 2020.

[231] T. Liu, S. He, S. Huang, D. Tsang, L. Tang, J. Mars, and W. Wang, "A Benchmarking Framework for Interactive 3D Applications in the Cloud," *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2020. [Online]. Available: http://dx.doi.org/10.1109/MICRO50266.2020.00076

[232] X. Liu, C. Vlachou, F. Qian, C. Wang, and K.-H. Kim, "Firefly: Untethered Multi-user VR for Commodity Mobile Devices," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, July 2020. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/liu-xing pp. 943–957.

[233] K. Boos, D. Chu, and E. Cuervo, "FlashBack: Immersive Virtual Reality on Mobile Devices via Rendering Memoization," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2906388.2906418 p. 291–304.

[234] Y. Li and W. Gao, "MUVR: Supporting Multi-User Mobile Virtual Reality with Resource Constrained Edge Cloud," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 1–16.

[235] C. Xie, X. Li, Y. Hu, H. Peng, M. Taylor, and S. L. Song, "Q-VR: system-level design for future mobile collaborative virtual reality," in *Proc. 26th ACM Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 587–599.

[236] Y. Bao, T. Zhang, A. Pande, H. Wu, and X. Liu, "Motion-Prediction-Based Multicast for 360-Degree Video Transmissions," in *2017 14th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2017, pp. 1–9.

[237] M. Dasari, A. Bhattacharya, S. Vargas, P. Sahu, A. Balasubramanian, and S. R. Das, "Streaming 360-Degree Videos Using Super-Resolution," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 1977–1986.

[238] X. Hou, S. Dey, J. Zhang, and M. Budagavi, "Predictive View Generation to Enable Mobile 360-Degree and VR Experiences," in *Proceedings of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network*, ser. VR/AR Network '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3229625.3229629 p. 20–26.

[239] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, "Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3241539.3241565 p. 99–114.

[240] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A Fully Integrated Energy-Efficient Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones," in *2018 IEEE Symposium on VLSI Circuits*, 2018, pp. 133–134.

[241] R. Pinkham, A. Berkovich, and Z. Zhang, "Near-Sensor Distributed DNN Processing for Augmented and Virtual Reality," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 663–676, 2021.

[242] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "imap: Implicit mapping and positioning in real-time," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6229–6238.

[243] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "NICE-SLAM: Neural Implicit Scalable Encoding for SLAM," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[244] D. Azinović, R. Martin-Brualla, D. B. Goldman, M. Nießner, and J. Thies, "Neural RGB-D Surface Reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[245] D. Cavalcanti, J. Perez-Ramirez, M. M. Rashid, J. Fang, M. Galeev, and K. B. Stanton, "Extending Accurate Time Distribution and Timeliness Capabilities Over the Air to Enable Future Wireless Industrial Automation Systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1132–1152, 2019.

[246] R. Singh, M. Huzaifa, J. Liu, A. Patney, H. Sharif, Y. Zhao, and S. Adve, "Power, Performance and Image Quality Tradeoffs in Foveated Rendering," in *Submission*, 2022.

[247] H. Sharif, Y. Zhao, M. Kotsifakou, A. Kothari, B. Schreiber, E. Wang, Y. Sarita, N. Zhao, K. Joshi, V. S. Adve, S. Misailovic, and S. Adve, "ApproxTuner: A Compiler and Runtime System for Adaptive Approximations," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3437801.3446108 p. 262–277.

[248] J. Alsop, M. D. Sinclair, and S. V. Adve, "Spandex: A Flexible Interface for Efficient Heterogeneous Coherence," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00031 p. 261–274.

[249] R. Komuravelli, M. D. Sinclair, J. Alsop, M. Huzaifa, M. Kotsifakou, P. Srivastava, S. V. Adve, and V. S. Adve, "Stash: Have Your Scratchpad and Cache It Too," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2749469.2750374 p. 707–719.

[250] M. Pellauer, Y. S. Shao, J. Clemons, N. Crago, K. Hegde, R. Venkatesan, S. W. Keckler, C. W. Fletcher, and J. Emer, "Buffets: An Efficient and Composable Storage Idiom for Explicit Decoupled Data Orchestration," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3297858.3304025 p. 137–151.

[251] M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, "The Accelerator Store: A Shared Memory Framework for Accelerator-Based Systems," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, Jan. 2012. [Online]. Available: https://doi.org/10.1145/2086696.2086727

[252] N. Pereira et al., "ARENA: The Augmented Reality Edge Networking Architecture," in *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2021.