FPGA-Roofline: An Insightful Model for FPGA-based Hardware Accelerators in Modern Embedded Systems

Moein Pahlavan Yali

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Patrick R. Schaumont, Chair
Thomas L. Martin
Chao Wang

December 10, 2014
Blacksburg, Virginia

Keywords: Embedded Systems, FPGA, Hardware Accelerator, Performance Model

FPGA-Roofline: An Insightful Model for FPGA-based Hardware Accelerators in Modern Embedded Systems

Moein Pahlavan Yali

(ABSTRACT)

The quick growth of embedded systems and their increasing computing power has made them suitable for a wider range of applications. Despite the increasing performance of modern embedded processors, they are outpaced by computational demands of the growing number of modern applications. This trend has led to emergence of hardware accelerators in embedded systems. While the processing power of dedicated hardware modules seems appealing, they require significant effort of development and integration to gain performance benefit. Thus, it is prudent to investigate and estimate the integration overhead and consequently the hardware acceleration benefit before committing to implementation. In this work, we present FPGA-Roofline, a visual model that offers insights to designers and developers to have realistic expectations of their system and that enables them to do their design and analysis in a faster and more efficient fashion. FPGA-Roofline allows simultaneous analysis of communication and computation resources in FPGA-based hardware accelerators. To demonstrate the effectiveness of our model, we have implemented hardware accelerators in FPGA and used our model to analyze and optimize the overall system performance. We show how the same methodology can be applied to the design process of any FPGA-based hardware accelerator to increase productivity and give insights to improve performance and resource utilization by finding the optimal operating point of the system.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Modern embedded applications have created a necessity for more powerful embedded platforms. In order to reduce time-to-market, flexible platforms are increasingly in favor for modern embedded applications. Therefore, software-based implementations are getting more popular in embedded world. Hardware acceleration is a technique to improve software performance by offloading a part of computation to a dedicated hardware module. To meet the performance requirements, hardware acceleration is sometimes used in embedded systems design. A successful hardware acceleration is the one that improves the performance significantly while does not dramatically increase the cost of design, development and operation of the system e.g. power consumption, price, production time, etc. Hardware-software integration is usually a time-consuming and difficult task. Committing to implementation of a hardware accelerator needs to be decided based on realistic expectations of outcome. Such expectations can be achieved by having an insightful model of the system that can estimate the performance bounds of the integration. The motivation of this work is to create such model. For example, assume that we have a software implementation of a cryptographic signature algorithm on an embedded processor as a part of our system. Assume that this

piece of software can perform 100 signatures per second which does not satisfy our performance requirement of 150 signatures per second. We find a hardware accelerator that can perform 200 signatures per second. If we decide to accelerate our software implementation using this hardware accelerator, we need to choose a communication method to integrate the accelerator into the system. The integration options may vary depending on the platform, but some communication overhead is definite. Every communication scheme bears some deficiency caused by various factors such as synchronization, latency, driver, etc. Hence, choosing a fast accelerator does not guarantee the desired performance boost by itself. Due to the software-hardware integration overhead, the obtainable performance on the processor is guaranteed to be less than the peak performance of the hardware module. Estimating this overhead is not usually easy, especially in a modern embedded system. Modern embedded processors typically run on a complex operating system and use complex interfacing techniques. Assume that the accelerator in question is integrated to the processor through a PCI Express bus. There are so many factors that can affect the overhead of integration such as the PCIe hardware module used on each side, the PCIe configuration on each side, the device driver running on the CPU, the application on the CPU, operation system overhead, etc. This work aims to simplify the performance estimation of hardware acceleration by introducing a insightful model that can abstract the system into a visual diagram and help the designer and developer to understand the bottlenecks of the system. The rest of this chapter extends the discussion about hardware acceleration in modern embedded systems and why our model is a useful asset.

1.2 Modern Embedded Systems

An embedded system is a combination of computer hardware and software—and perhaps additional parts, either mechanical or electronic—designed to perform a dedicated function. [5] Indeed, the definitive characteristic of an embedded computer is that it is designed around a specific application, as opposed to general purpose computers. The conventional wisdom

in embedded system design is that typical embedded computers have other distinctive properties compared to general purpose ones e.g. low cost, low power consumption, small size, etc. that limits their processing power and interfacing options. This comparison remains valid as the world of computing grows. Embedded systems are no exception to Moore's Law. Hence, as other computing systems such as supercomputers or personal computers continue to scale, so do embedded systems. As a result, new applications with increasing computational demands are entering the embedded realm. Mobile industry is a substantial evidence to this trend. Smartphones as modern high-end embedded computers have created an increasing demand of processing power within the design constraints of highly portable embedded systems. Modern embedded processors, although powerful, are still outpaced by these increasing demands. This significant growth in the embedded market boosts the research and development efforts in this area and leads to rapid production of new technologies. This trend has created a new ecosystem in embedded system design. Microcontrollers, small memory, legacy serial connectivities, and standalone software are giving their place to multicore processors, large memory, complex buses, and operating systems. Consequently, embedded designers and developers have an imminent need for new productive methods to design, develop and optimize more efficiently.

1.3 Hardware Acceleration

The primary challenge of modern embedded systems is to address the rapidly increasing demand of processing power in new applications. This is in fact the fundamental challenge of computing industry as a whole. HPC industry has been dealing with a similar challenge for years. Figure 1.1 demonstrates how the increasing demands of modern applications have outpaced the conventional processor's ability to deliver that scales by Moore's Law. As shown in Figure 1.1, this trend has created *the technology gap* between the supply and demand of computational power. A tremendous amount of research has been done on performance analysis and optimization to maximize the computational leverage of existing computer architectures

and aim for scalable architectures in the future. An informal observation of the *Pareto Principle* (also known as 80-20 rule) in performance analysis literature suggests that 80% of the processing time is spent in the 20% of the code. This insight motivates profiling in order to detect that 20% and speed it up to have the most optimization gain. Hardware acceleration is one of the common solutions to achieve the desired speed up. Hardware accelerators have been in use by supercomputers and high performance computing industry to address the technology gap in question. Due to the inflexibility of the hardware, application-specific hardware is usually not a cost-efficient option in general purpose computers. However, in some cases it is proven to be efficient to use customized hardware to speed up the performance bottleneck of the software. For example, most modern ethernet cards support TCP checksum offloading as it is a costly part of TCP packet processing. The proof of efficiency may depend on several aspects of the system e.g. production cost, power consumption, performance gain factor, etc. and requires case by case in-depth profiling and investigation. Obviously, it is more likely to see hardware acceleration in a supercomputer rather than a desktop computer, as supercomputers are more cost tolerant and hence have significantly different standards of efficiency. A specific hardware accelerator might be too costly to be used in a personal desktop computer or a low power handheld device, but very efficient if used in a supercomputer. Therefore, HPC industry has been the primary consumer of hardware accelerators in the past. With the emergence of modern embedded systems that are typically more *heavyweight* in contrast to their *lightweight* predecessors and the highly demanding applications in the embedded world, hardware accelerators have begun their presence in embedded systems. Since hardware acceleration is a relatively new technique in embedded systems, the methods for design, development, analysis and optimization of hardware accelerators for embedded systems are not as evolved as they are in HPC industry. By looking at the existing works on this topic in HPC literature, we investigate if it is feasible to adopt their solutions in the embedded system design or look for insights and directions to develop similar solutions. However a similar trend in HPC and embedded computing has caused the emergence of hardware acceleration, there are fundamental differences between

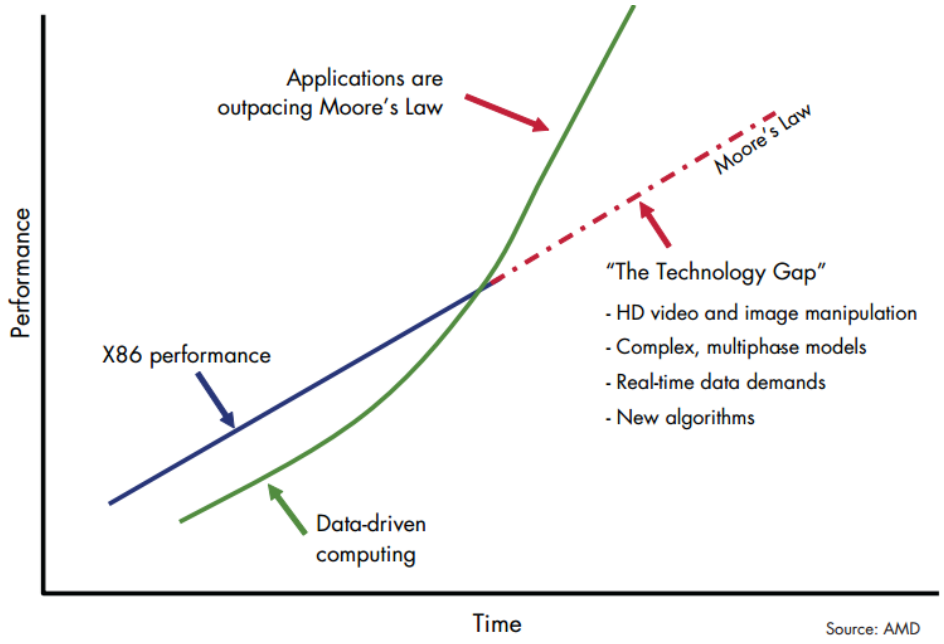


Figure 1.1: Industry Landscape: The Technology Gap [2]

Source: Accelerating high-performance computing with FPGAs. Altera Corporation, October 2007. [Online; accessed: 2014-11-20 at http://www.altera.com/literature/wp/wp-01029.pdf], Used under fair use, 2014

HPC and embedded systems. It is important to understand these differences as we study the literature.

1.4 Hardware Acceleration in High Performance Computing vs. Embedded Computing

The fundamental difference between high performance computers and embedded computers is in their objective. The definitive property of embedded systems is that they are specific to a task. They are designed and developed around a specific application and they are usually optimized to address the needs of that application. This is not true for supercomputers. Supercomputers are essentially a cluster of powerful general purpose processors and they are

not designed for a specific application. Rather, they need to address the needs of a variety of computationally heavy applications. This fundamental difference is reflected all over the literature of these two communities. HPC community is always concerned by flexibility, scalability and reusability. How a proposed method/tool can be scaled and used across different applications or through different generations is an essential metric of efficiency. Therefore, hardware acceleration solutions that could offer some levels of flexibility and scalability are more favorable in HPC industry. Namely, Field Programmable Gate Arrays (FPGA) and Graphical Processing Units (GPU) are the primary solutions with such quality. There is an immense amount of work in HPC literature for research and development of scalable and flexible methods and tools to leverage FPGA/GPU co-processors in supercomputers [10, 6, 8]. To address the needs of HPC market, FPGA and GPU vendors introduced technologies and tools for efficient hardware acceleration. [2] is an example of such efforts by Altera to justify their products as efficient hardware acceleration solution for HPC market. They discuss how their platforms and development environment capture the four primary needs of HPC industry: Power, Performance, Price and Productivity.

The technology gap is not limited to high performance computing. It is now an evident phenomena in embedded computing. The new demands of the embedded market is pushing the industry to provide solutions to fill this gap. In a similar trend, hardware accelerations are now introduced to embedded systems. Development of methods and tools for hardware acceleration in modern embedded systems is an imminent need. Due to the fundamental difference between embedded systems and supercomputers, the existing methods and tools used in HPC industry are not useful for development of embedded system. Such methods are designed to be scalable and flexible to be useful for a variety of applications, and the ideal practice of tools is to automatically generate a hardware accelerated solution from an existing software implementation. On the contrary, embedded systems are usually designed and built around a specific applications. The application does not need to be *mapped* to the platform, as the platform is essentially built or configured to run the application. In this development environment, the application engineer needs to identify the parts of application

that are worthy of being accelerated by hardware, then develop or find the corresponding hardware module, and finally —and most importantly— integrate the hardware module into the system. To go through this process, an application engineer needs to have the technical knowledge to integrate the accelerator to the system. This is still costly and time-consuming. So, the application engineer needs to have a clear idea of the performance gain that can be obtained by hardware acceleration, before committing to the process of implementation and integration. The peak performance of hardware modules can be usually found in their specification, but those values are reported regardless of the system into which the hardware is being integrated. Only a portion of the peak performance will be delivered to the system after integration due to the overhead of communication. Being able to estimate this value before implementation gives the application engineer a competitive edge on choosing the right accelerator and/or deciding if a process is worthy of being accelerated by hardware. This is an important productivity concern regarding hardware acceleration. Automated tools to do such process are very limited, even in HPC industry. Plus, it is not always desirable to develop and use such automated tools due to the more strict design constraints that mandate manual tuning and optimization. The primary objective of this work is to address this productivity concern. We propose a model and methodology that helps designers and developers to have realistic expectation of the system and rapidly analyze the system and its bottlenecks. We explain the details of our methodology in Chapter 3.

1.5 FPGA-based Hardware Accelerators

FPGAs as reconfigurable hardware platforms are a great candidate for hardware acceleration. The flexibility offered by FPGA reconfigurability —however inferior to software flexibility— has made them an appealing hardware acceleration solution in HPC industry. Their cost efficiency compared to ASIC solutions and their power efficiency compared to microcontrollers have made FPGAs a widely used hardware platform in embedded systems. In modern scheme of embedded system design, FPGAs offer a potent solution to equip embedded processors

with extra processing power. FPGA vendors are pushing their technology to justify FPGAs as an efficient solution of hardware acceleration in embedded systems. For example, Xilinx Zynq family and Altera Arria V SoC FPGAs, two recent platform families that integrate a hardcore embedded processor with a reconfigurable fabric on the same chip, bring a new design dimension to embedded systems, and a new challenge to the embedded system designer. As appealing as FPGAs appear for hardware acceleration, they would be in limited favor if they require significant development effort. Productivity is arguably the most challenging obstacle for FPGAs to qualify as ideal hardware accelerators, because performance gain is beneficial only if it could be achieved in a limited design time.

1.6 FPGA-Roofline in a nutshell

In this work, we propose FPGA-Roofline, a visual model that provides a high level view of the system and offers helpful insights for performance estimation, optimization and resource utilization in FPGA-based hardware accelerated systems. This model is inspired by the Roofline model [13] that presents an insightful performance model for modern processor architectures based on 'bound and bottleneck' analysis. We discuss this model in detail in Chapter 3. FPGA-Roofline is created by a similar approach, but adapted to capture the fundamentally different challenges in embedded systems. FPGA-Roofline captures both communication and computation aspects of a hardware accelerator in one model. This gives a realistic expectation of the system to designers and developers that can lead to significant boost in productivity. In Section 1.4 we talked about the productivity challenge of hardware acceleration in design, implementation and integration stages. FPGA-Roofline equips the engineer with high level view of the system and its main bottlenecks and provides insightful information about the system to speed up the production process. Using FPGA-Roofline, one can model all core components of the system into an easy-to-understand diagram. This model helps the developer to understand the system better and have realistic expectations before committing to implementation. By identifying performance bottlenecks, whether in

communication or computation, FPGA-Roofline provides insights and directions to optimize the system performance and resource utilization. Therefore, FPGA-Roofline primarily addresses productivity and performance. Moreover, we will show how our model can be also insightful to address efficiency. Having a high level view of the system, one can find the optimal operating point of the system and adjust the design parameters to have an efficient system that maximizes the resource utilization while delivering the maximum performance. Chapter 3 is dedicated to the details of our model. We explain the principal concepts behind our model and how it can be formed and used in a typical development scenario.

Chapter 2

Background

2.1 Platform-based Design Methodology

In Chapter 1 we discussed the similarities and differences between embedded computing and high performance computing. Modern embedded systems are facing a challenge that has been the main focus in HPC industry and computer architecture community for years. It is not groundless to assume that there is a better consensus about performance-oriented challenges in HPC and computer architecture community than there is currently in embedded community. With that in mind, we explored the literature of HPC and computer architecture to hopefully find clues or insights that can lead to similar solutions in embedded systems. It was no surprise that most works shared a similar point of view based on the current tools and methods in HPC community. Their focus was mainly in developing automated tools that can convert a full software implementation to a hardware accelerated implementation. The evaluation methods were also mainly concerned by how their tools and methods works across a variety of applications. For example, M. Meswani et al. [4] develop a modeling tool that predicts the performance of HPC application if accelerated by GPU or FPGA. Their process of modeling includes extracting computational patterns from the code and co-simulation of the extracted data along with the application traces and machine profiles

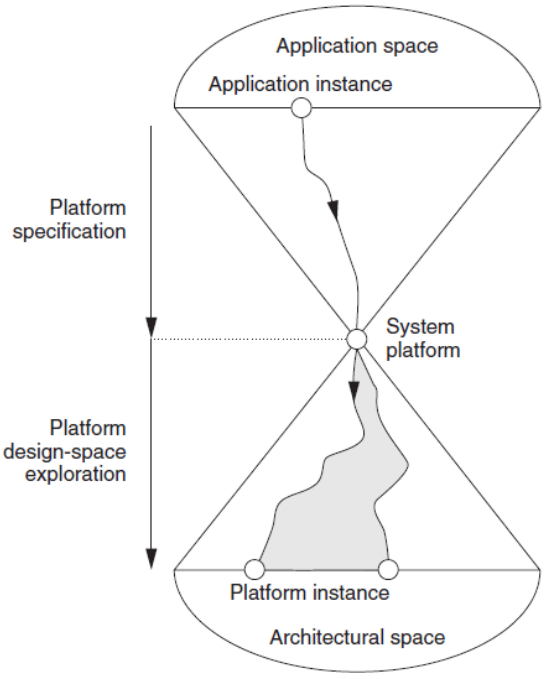


Figure 2.1: System platform layer and design flow [9]

Source: A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. Design and Test of Computers, IEEE, 27(2):23-33, December 2006, Used under fair use, 2014

by using some other existing tools and frameworks. Most of the performance modeling work is based on such statistical and stochastic methods. This kind of approach is not common —if not nonexistent— in the current scene of embedded system design. There are no such tools as simulation-based performance prediction frameworks or machine profiles to improve productivity.

The absence of such tools and methods in embedded system design is mainly because the traditional design methodology in embedded systems was not *platform-based*. A. Sangiovanni-Vincentelli and G. Martin [9] back in 2002 pointed out the importance of platform-based design methodology in the future of embedded systems. The increased complexity in embedded applications, constantly evolving specifications and expensive hardware-manufacturing

cycles forced embedded designers to consider intrinsically flexible implementations. Consequently, embedded processors started to become more popular. The negative side of this movement was that PC-like software development techniques were not designed to consider the requirements of embedded systems. The software world had paid little attention to hard constraints on issues like safety verification, softwares reaction speed, memory footprint, and power consumption. Hence, using the traditional methods of software development created a crisis —as they call it— in embedded software design. They draw a theoretical roadmap to a platform-based design methodology for the future of embedded system design. Their basic idea is demonstrated in Figure 2.1. The system platform effectively decouples the application development process (the upper triangle) from the architecture implementation process (the lower triangle). A system designer maps an application onto the abstract representation, choosing from a family of architectures to optimize cost, efficiency, energy consumption, and flexibility. The vertex where the two cones meet represents the combination of the API and the architecture platform. To choose the right architecture platform, we need to export to the API level an execution model of the architecture platform that estimates its performance. This is an ideal development environment with high productivity and efficiency. Despite the significant improvements in design methodologies in the last decade, the reality of embedded system design is still far from this ideal. In most cases, embedded engineers need to use ad-hoc procedures and case-specific considerations. The concept of *platform* is usually limited to the target hardware and its development tools rather than a high level API with profiling tools and execution models for performance prediction. As a result, the productivity gap in embedded software development is far larger than general purpose software development. This results in an increase in the time-to-market and forces developers to make compromises with regard to other design constraints.

The lack of platform-based embedded software development ecosystem makes it infeasible to have an embedded version of high level automated tools such as the performance prediction framework for hardware acceleration in high performance computing applications presented by [4]. Currently, the usual scheme of hardware acceleration in embedded systems is to

implement and evaluate. It is only after implementation that the efficiency of acceleration can be evaluated. Also, every optimization or tuning can be evaluated only after implementation. The outcome is a very slow design flow with time-consuming iterations. P. Schaumont and I. Verbauwhede [11] found that a key challenge in teaching embedded system design is the lack of a general consensus in how an optimization can affect the performance. The performance of the final implementations of several students, who were given the same project baseline, can be clustered into a few distinguished clusters. Each cluster corresponds to a different set of optimizations that students could find and apply to their project. To explore these optimization opportunities and implement them, students had to heavily rely on their creativity. Therefore, those who were more creative and persistent explored more and better. Another important asset in optimization is to know when to stop optimizing. In other words, knowing a performance upper-bound can help in deciding whether an optimization is good enough or not. A performance model that provides insightful information about the bottlenecks of the system and its performance upper-bounds can speed up and refine the process of design space exploration. This results in a more productive design flow. FPGA-Roofline is an insightful model which equips the designer and developer with such information for FPGA-based hardware acceleration in embedded system. It creates a high level understanding of the system and the bottlenecks of hardware acceleration that can be used in different steps of the design, development and optimization. FPGA-Roofline is well suited for platform-based design methodology. Each platform can be profiled by a platform expert only once, and the extracted specifications can be used by developers to create a performance model of the platform for their application. Then, the developer can decide if the platform meets the application performance requirements. Moreover, the created model can help the developer to recognize the bottlenecks of the platform with regard to the application and consequently design, develop, and optimize more efficiently. This is fundamentally analogous to the process of HPC tools that simulate application traces against machine profile to predict performance, yet it uses a much simpler methodology that suits the embedded system design environment. The details of FPGA-Roofline are discussed in

2.2 The Roofline Model

The original idea of FPGA-Roofline was inspired by the Roofline Model [13]. The roofline model is an insightful performance model for multicore processor architectures. Unlike most performance models in HPC literature, roofline model does not use stochastic or statistical techniques. "Stochastic analytical models and statistical performance models can accurately predict program performance on multiprocessors but rarely provide insight into how to improve the performance of programs, compilers, and computers and can be difficult to use by nonexperts. An alternative, simpler approach is bound and bottleneck analysis. Rather than try to predict performance, it provides valuable insight into the primary factors affecting the performance of computer systems. In particular, the critical influence of the system bottleneck is highlighted and quantified." [13] This simple approach towards performance modeling is suitable for embedded system design. As described in Section 2.1, current environments for embedded system design is less sophisticated than general purpose and high performance computing. Therefore, simple approaches like roofline model can be a guideline for similar ideas in embedded system design. We have used the idea of roofline model to develop FPGA-Roofline which is specifically concerned with FPGA-based hardware acceleration. We believe that "bound and bottleneck" approach is interestingly useful for this purpose and we show its promises in Chapter 3.

We provide a detailed definition of our model in Chapter 3. However knowing the original Roofline model is not necessary to understand our model, it can be highly beneficial to understanding the principal concepts. The roofline model is based on the fact that for the foreseeable future, off-chip memory bandwidth will often be the constraining resource in system performance [7]. Hence, they want a model that relates processor performance to off-chip memory traffic. Toward this goal, they define the term "operational intensity"

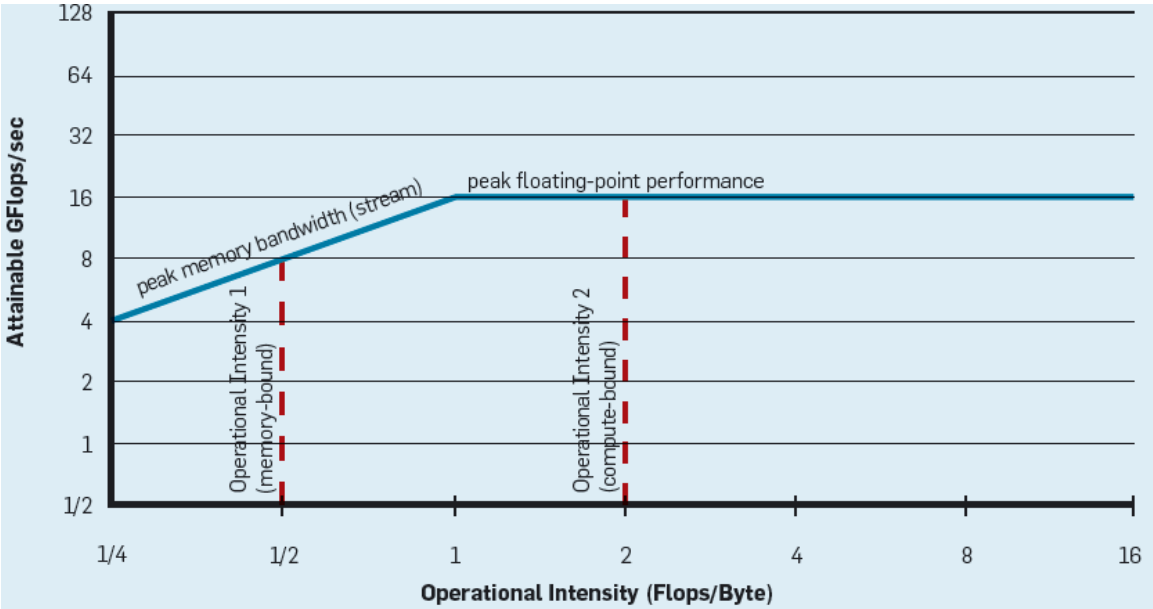


Figure 2.2: Roofline model for AMD Opteron X2 [13]

Source: S. Williams, A.Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. Communications of the ACM, 52(4):65-76, April 2009, Used under fair use, 2014.

to mean operations per byte of DRAM traffic. That is, they measure traffic between the caches and memory rather than between the processor and the caches. The Roofline model ties together floating-point performance, operational intensity, and memory performance in a 2D graph. Peak floating-point performance can be found through hardware specifications or microbenchmarks. Memory performance is found through running optimized microbenchmarks designed to determine sustainable DRAM bandwidth. Figure 2.2 outlines the model for a 2.2GHz AMD Opteron X2 model 2214 in a dual-socket system. The graph is on a log-log scale. The Y-axis is attainable floating-point performance. The X-axis is operational intensity, varying from 0.25 Flops/DRAM byte-accessed to 16 Flops/DRAM byte-accessed. The system being modelled has peak double precision floating-point performance of 17.6 GFlops/sec and peak memory bandwidth of 15GB/sec from our benchmark. One can plot a horizontal line showing peak floating-point performance of the computer. The actual floatingpoint performance of a floating-point kernel can be no higher than the horizontal line, since this line is the hardware limit. Since the X-axis is Flops per Byte and the Y-axis is GFlops/sec, gigabytes per second (GB/sec) —or (GFlops/sec)/(Flops/Byte)— is just a line of unit slope in Figure 2.2. Hence, we can plot a second line that bounds the maximum floating-point performance that the memory system of the computer can support for a given operational intensity. This formula drives the two performance limits in the graph in Figure 2.2:

$$\text{Attainable GFlops/sec} = \min \begin{cases} \text{Peak Floating-Point Performance} \\ \text{Peak Memory Operational Bandwidth} \times \text{Intensity} \end{cases}$$

For a given kernel, we can find a point on the X-axis based on its operational intensity. If we draw a vertical line (the red dashed line) through that point, the performance of the kernel on that computer must lie somewhere along that line. The horizontal and diagonal lines give this bound model its name. The Roofline sets an upper bound on performance of a kernel depending on the kernels operational intensity. If we think of operational intensity as a column that hits the roof, either it hits the flat part of the roof, meaning performance is compute-bound, or it hits the slanted part of the roof, meaning performance is ultimately

memory-bound. Note that the ridge point (where the diagonal and horizontal roofs meet) offers insight into the computers overall performance. The x-coordinate of the ridge point is the minimum operational intensity required to achieve maximum performance. If the ridge point is far to the right, then only kernels with very high operational intensity can achieve the maximum performance of that computer. If it is far to the left, then almost any kernel can potentially hit maximum performance. [13]

More details and evaluation of the roofline model can be found in the original paper of the Roofline model [13]. In Chapter 3, we define our inspired model for embedded hardware accelerators from scratch in a bottom-up fashion. In Section 5.1, we compare our model with the original Roofline and discuss their differences and similarities.

2.3 Little's Law in Computation and Communication

The principal concepts of the Roofline model can explained by Little's Law [3]. In queuing theory, Little's Law is theorem by John Little, which states:

The long-term average number of customers in a stable system L is equal to the long-term average effective arrival rate, λ, multiplied by the average time a customer spends in the system, W; or expressed algebraically: $L = \lambda W$.

This simple theorem can be applied to any system or sub-system including computer systems. A computer science interpretation of this theorem can be stated as following:

$$Concurrency = Bandwidth \times Latency$$

Concurrency may apply to different computing entities such as concurrent data or concurrent operations. Hence, we define concurrency for a computation device and a communication device separately. In a data communication link with latency L and bandwidth B, the largest amount of data concurrently being transferred in the link is B $\times$ L. In other words, this is the least amount of concurrent data that should be provided to the link in order to have it fully

utilized. This applies to a memory subsystem with known bandwidth and latency values. Similarly, for a computation device with latency L and the operating rate B (number of operations per time unit is interpreted as the device bandwidth), the number of concurrent operations being computed in the device is B $\times$ L. Note that the unit of concurrency is dependent on the units of B and L. For a typical communication link this unit would be Bytes. For a computation device it depends on the operation done by the device. For example, HPC community usually use FLOP as the unit of operation in high performance computing systems due to the domination of floating-point operations in most HPC kernels. Therefore, in a typical HPC system, the unit of communicational concurrency is Bytes and the unit of computational concurrency is FLOP. This is the principal concept behind the definition of "operational intensity" in the Roofline model in Section 2.2. The unit of operational intensity is FLOP/B which relates the computational and communicational concurrency values of the kernel as a ratio. The ridge point on the Roofline model of a platform basically highlights the minimum value for this ratio that makes the kernel compute-bound on that platform.

In a hardware-software integration scheme, this can be used as the basis of analysis. If the hardware component is not able to process enough data concurrently, then the link is underutilized. On the other hand, if the hardware can process more than this value the link is fully utilized; however this makes the hardware wait for the link once in a while. Hence, the Little's Law can be used to find a balanced point between the computation intensity and communication intensity of the system. On this balance point, the device provides just enough data to the link to be fully utilized. As a result, just enough bandwidth is provided to the computing element to send the data over the link without waiting for the link availability. This concept is illustrated in Chapter 3.

Chapter 3

FPGA-Roofline

In this chapter, we explain the principles of our model, how it works, and how to build it by abstracting the platform and application.

3.1 Host-Device Model

A typical hardware acceleration consists of three main components: host processor, coprocessor (device), and communication link. Respectively, we will refer to these three components as *host*, *device* and *link*. The modeling is minimal but complete: host and device exchange data through the link. The first step of using the model is to identify these three elements in the system. Figure 3.1 shows how a typical FPGA-based hardware acceleration can be modelled as host-device communication. An application running on the host processor needs

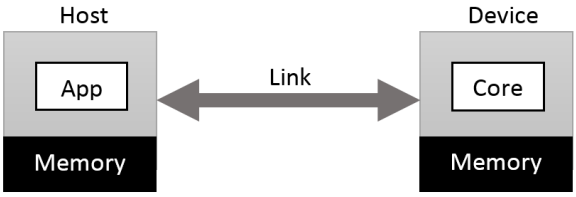


Figure 3.1: Host-Device model in FPGA-based hardware acceleration

to leverage the computational power of a core on FPGA. The grey margin on the host block represents the software environment e.g. operating system, device driver, etc. The core refers to the computational logic in the FPGA. The core is the unalterable part of the hardware that provides the computing power that we intend to deliver to the software. The grey margin on the device side represents the hardware logic wrapping the core e.g. controller, DMA unit, etc. Each side has a local memory which provides fast local access. The link is used by each peer to transfer data from its local memory to the other peer's memory and vice versa.

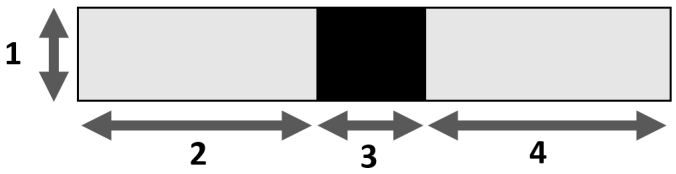
3.2 Communication Model

Different platforms use different means of communication. To be able to generalize the communication scheme of hardware acceleration, we make an important assumption:

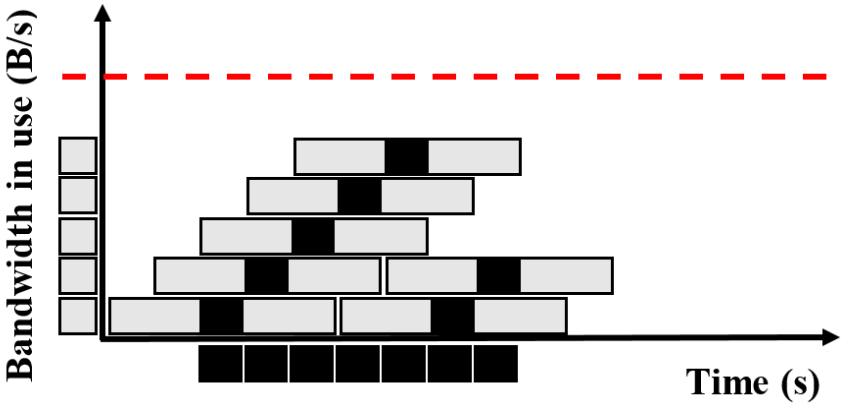
The device core is used to accelerate an identifiable operation that receives a known amount of input data and computes a known amount of output data in a known amount of time.

In other words, it is a necessity for building the model to know the amount of I/O data and the latency of the core to complete a single task which we will refer to as *unit operation*. We believe that this restriction does not limit the scope of applications that can be supported. Furthermore, in case we have non-manifest bounds in terms of computational load or communication load, we propose that our model would work with the known upper-bound of these quantities. The device can be *invoked* by the host to complete one or more unit operations. An *invocation* consists of three steps: transferring input data from host to device; computation on device; and transferring output from device to host. In general the number of unit operations per invocation can be more than one, although in many cases application requirements may force this parameter to one.

To have an easy-to-understand representation of this communication scheme, we use the diagram shown in Figure 3.2.b. Figure 3.2.a shows the building block of this diagram. Each



a. Invocation block



b. Invocation ordering diagram

Figure 3.2: Visual model of communication

block represents a single invocation. We assume there are no data dependencies between subsequent accelerator invocations. The width of the block represents the time needed to complete the invocation. The block is divided to three sections in width: the black section is representing the computation time of the core(s) and the grey sections represent the input/output data transfer time. The area of grey sections on the left and right respectively represent the data payload needed to be transferred as input and output to complete the invocation. Consequently, the height of the block will represent the bandwidth being used by the invocation. The blocks are arranged on a two- dimensional diagram. The horizontal axis represents time and the vertical axis represents the effective data bandwidth in use. A horizontal dashed line represents the link bandwidth limit which is the upper limit for the vertical axis. The projection of black sections are drawn on the horizontal axis while the projection of grey sections are drawn on the vertical axis. Although this diagram can be numerical, we do not use it for quantative purposes since its purpose is only to visualize a concept rather than any quantative analysis. Presumably, it is difficult to profile a hardware/software integrated system to build a numerical version of such diagram for analysis. We use this diagram to explain a concept rather than a solution. We use this communication model to visualize the ideal order of invocations in the system, given the following constraints in arranging the blocks:

1. Blocks cannot overlay, instead they pile up vertically. A larger pile of blocks means more concurrent invocations and more data in the link.

2. No vertical line must cross more than one black section of the blocks. Because at any given time, the device core can be responding only to one invocation.

3. The pile of blocks cannot exceed the bandwidth limit.

These three constraints in fact represent the constraints of the system. Any arbitrary ordering of invocations that does not violate them can potentially happen in the system. If we set our goal to maximize the number of unit operations completed in unit of time (throughput),

we can simply visualize an optimal order of invocations by adding the following constraint:

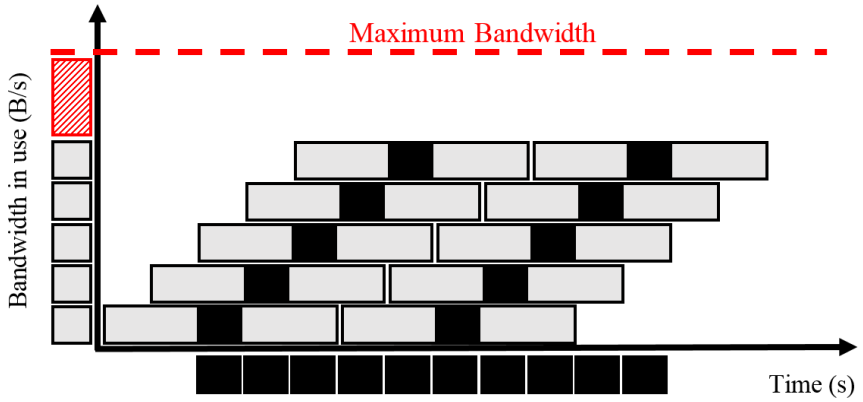
4. Taking constraints 1 to 3 into account, each block must be placed in the closest possible position to the left bottom corner of the graph.

This constraint enforces the most compact way of arranging the blocks which corresponds to the optimal ordering of invocations for maximized throughput of the system. Figure 3.3 shows two examples of achieving this optimal ordering by applying all four constraints. Figure 3.3.a shows an ordering in which the link bandwidth is always underutilized, but the device is continuously operational i.e. fully utilized. Figure 3.3.b shows a scenario in which the link bandwidth is fully utilized, however the device has some idle time and is underutilized. Resource underutilization is shown by red diagonal pattern on both figures. It is important to note again that both orderings are optimized for maximum throughput, yet in each case throughput is *bound* by a different resource. The first one is bound by the core throughput, while the second one is bound by the link bandwidth. This important observation is the foundation of our model.

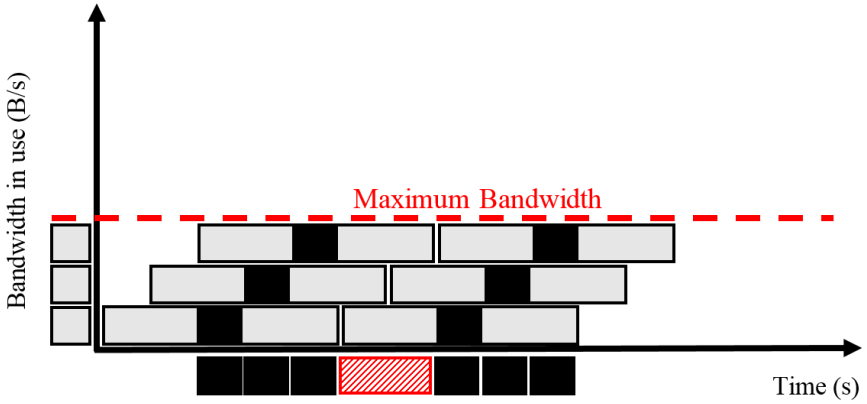
3.3 Operational Intensity

As observed in the last section, throughput of the invocations can be bound by either the bandwidth of the link or the throughput of the core. As the diagram suggests, the shape of the blocks has direct effect on which case happens in a particular system. A simple observation concludes that blocks with a larger grey section pile up and fill up the link bandwidth faster. This leads us to an important concept: The amount of data transfer in invocations is the key factor of determining the bottleneck of the system throughput. We define *operational intensity* based on this concept as follows:

Operational intensity is defined as the number of unit operation done in each invocation divided by the number of data bytes transferred in each invocation.



a. Bound by the core throughput



b. Bound by the link bandwidth

Figure 3.3: Invocation ordering optimized for maximum throughput

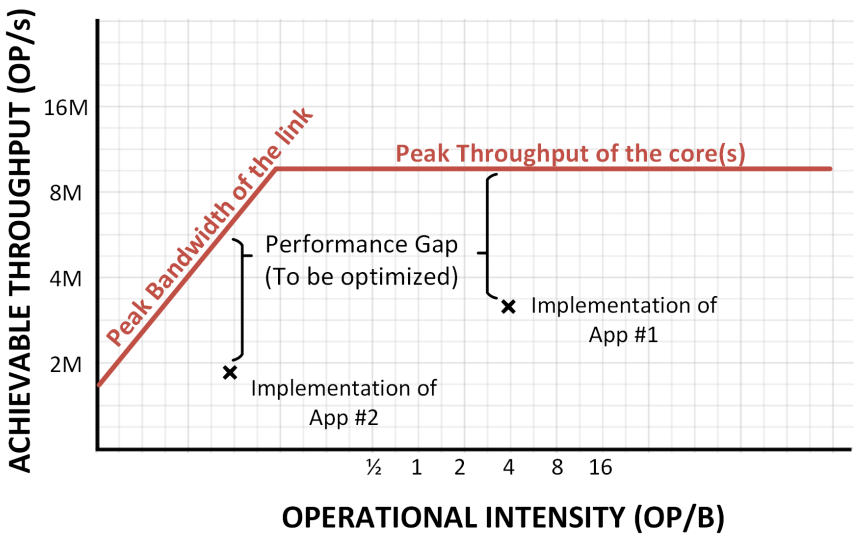
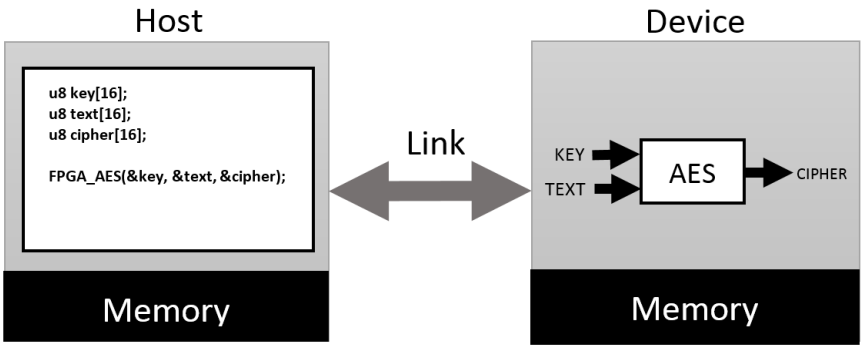
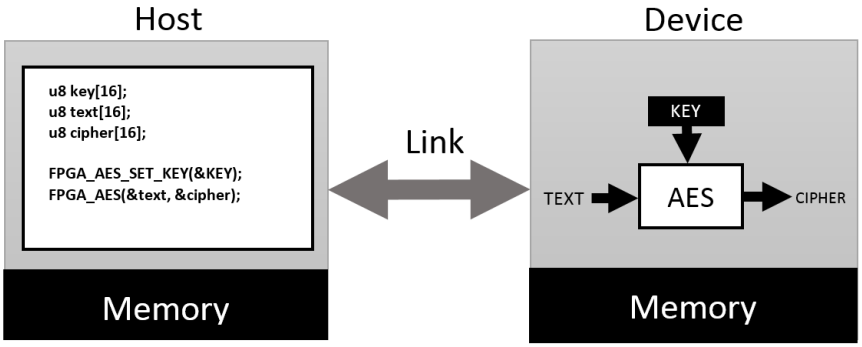


Figure 3.4: FPGA-Roofline Model

For further illustration we come up with the example shown in Figure 3.5. Advanced Encryption Standard (AES) is a well-known cryptographic algorithm. The basic version of AES uses a 128-bit key to encrypt a 128-bit block of plaintext and generate 128-bit of ciphertext. Hence, a typical AES encryption core has two 128-bit inputs and one 128-bit output which sums up to 48 bytes of I/O data for a single AES encryption. Knowing how the AES core works is not enough to determine operational intensity as 1/48 AES/B, since by definition we should know how the invocations work. Knowing the unit operation by itself is not enough for determining the operational intensity. Figure 3.5 illustrates this matter for our AES example. If each invocation consists of sending two 128-bit words as key and plaintext and receiving one 128-bit word as ciphertext, then the operational intensity would be 1/48 AES/B. However in case of many applications a single key can be used for a large amount of plaintext and gets replaced once in a while. In that case, it makes sense to design the hardware accelerator to save the key in a register rather than sending it with each invocation. Consequently, each invocation will only have 32 bytes of data transfer and the operational intensity will be 1/32 AES/B.



a. AES invocation with operational intensity of 1/48 AES/B



b. AES invocation with operational intensity of 1/32 AES/B

Figure 3.5: Integration of AES core with different values of operational intensity

3.4 Building the FPGA-Roofline Model

In this section, we have so far covered all the principal concepts necessary to understand our model. The idea behind this model is originally taken from [13] that presents a similar model for modern processor architectures. FPGA-Roofline is an adaption of the original roofline model for FPGA-based hardware accelerators. FPGA-Roofline is a visual model and is presented as a diagram. Figure 3.4 shows how this diagram is formed. The horizontal axis is operational intensity and its dimension is OP/B. The vertical axis is achievable throughput and its dimension is OP/s. Dividing the vertical dimension by the horizontal dimension gives us the dimension of the slope in this diagram which is B/s which is the same of link bandwidth. The roofline is formed as follows:

The slope of the diagonal part of the roofline is equal to the peak bandwidth of the link. The height of the horizontal part of the roofline is equal to the peak throughput of the core.

As we discussed earlier in Section 3.2 the system throughput is bound by either the peak throughput of the core or the peak bandwidth of the link. The roofline-shaped line is also created based on these two important parameters of the system and therefore is unique for the system. To form the FPGA-Roofline model, one only needs to know these two parameters. They can be usually found in the specifications of the system components. FPGA cores normally have known latency values in their specifications from which the peak throughput can be calculated for a given clock frequency. Bus protocols and bridges that are used for host-device communication also have known values of peak bandwidth. Thus, forming the FPGA-Roofline model for a given hardware accelerated system is straightforward. The third parameter to find out is the operational intensity which was discussed in the previous section. Operational intensity determines where we should mark down on the horizontal axis to find the corresponding value of vertical axis as maximum achievable throughput. If it lies to left side of the ridge point the bottleneck is the link peak bandwidth and if it lies to the right side of the ridge point the bottleneck is the core peak throughput We discuss advance analysis and insights that can be obtained with this model in Chapter 5.

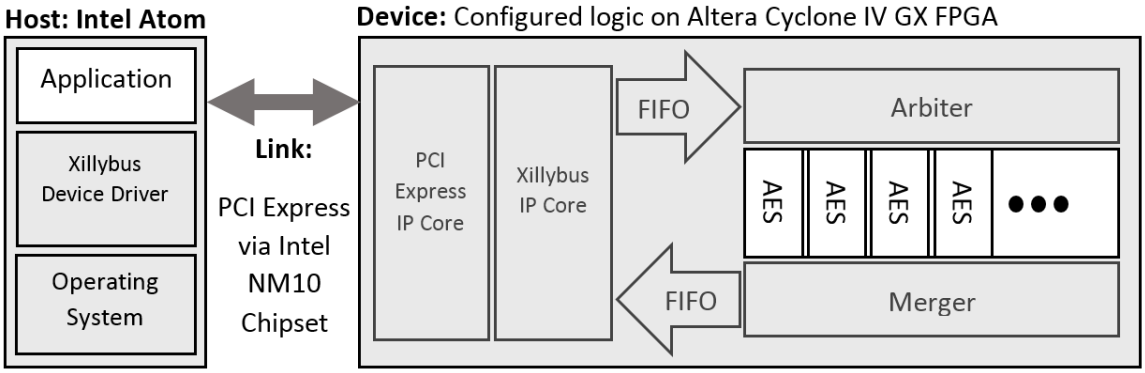


Figure 3.6: Block diagram of the implemented AES hardware accelerator

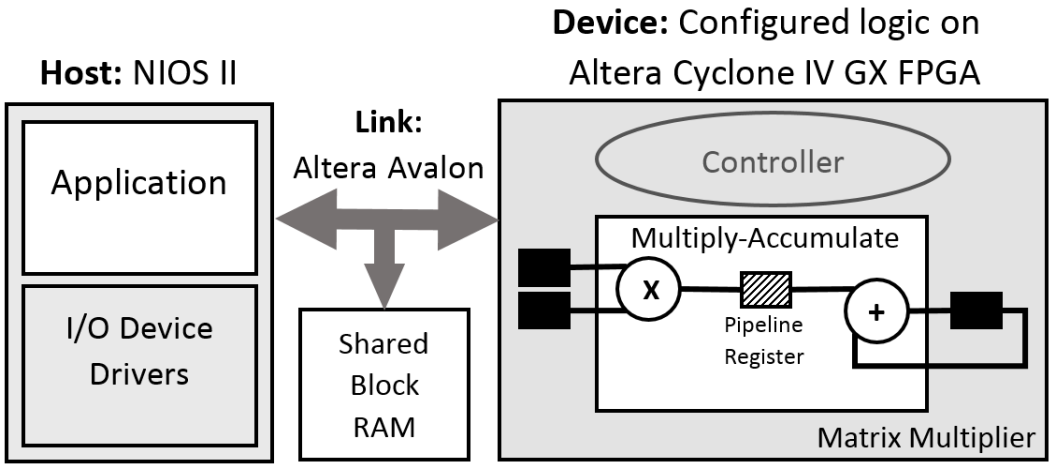


Figure 3.7: Block diagram of the implemented squared matrix hardware accelerator

Chapter 4

Evaluation

To evaluate our model and demonstrate its practicality, we have implemented two FPGA-based hardware accelerators. We have done a set of experiments on these implementation in order to reflect the capability and insightfulness of our model in design and analysis. In this section, we discuss these implementations and the result of the experiments.

4.1 Evaluation Platform

To choose an experimental platform, we had to make sure that we have a realistic representation of different schemes of hardware acceleration. Terasic DE2i-150 is a recent prototyping platform that integrates an Altera Cyclone IV GX FPGA and an Intel Atom N2600 dual-core processor on a single board along with numerous peripherals. An Intel NM-10 chipset provides connectivity between the FPGA and the processor through PCI Express 1x bus standard. The operating system running on the Atom processor is built by Yocto, a project sponsored by Intel for building custom versions of embedded Linux. We have used Yocto 1.5 standard build (no customization) called Dora (version 10.0.0) which is based on Linux 3.10.11 kernel. To leverage the PCI Express connectivity we have used Xillybus [1], an IP

core that works on top of the PCI Express IP core to provide an easy communication interface between the synthesized hardware in the FPGA and the software running in Linux or Windows. This combination of components provides a high-end platform for practical hardware acceleration. Moreover, synthesis of Altera NIOS softcore processor on the powerful Cyclone IV GX FPGA enables us to investigate system-on-chip hardware-software integration.

4.2 Implementations

Advanced Encryption Standard: The first application we use for evaluation is AES. Our AES FPGA core is based on the Verilog implementation of AES by R. Usselmann[12]. The latency of our AES core is 20 clock cycles. Doing one AES per invocation and using a single AES core as the accelerator narrows down the experiment to reflect only one specific scenario. Instead, we assume that an arbitrary application may need multiple executions of AES per invocation and also we may have multiple AES cores on the device side. Our implementation can be configured to handle different values of AES operations per invocation and leverage a configurable number of AES cores for computation. Therefore, we can form our model for different values of operational intensity and different values of the device peak throughput. The host application runs on the Atom processor and the host-device communication is done through the PCI Express connectivity. Figure 3.6 shows the block diagram of the device. Xillybus IP core provides FIFO interfaces on top of the PCI Express IP core. Inbound data gets buffered in a 128-bit register and an arbiter module feeds it into an available AES core. Each AES core buffers its output in a register and waits until the merger module reads it and feeds it into the outbound FIFO interface. AES cores are the computing resource and everything else is modelled as wrapping logic.

Squared Floating-Point Matrix: The second application is squaring a matrix. Matrix multiplication is a very common computational pattern. For simplicity, we have used squared matrix which is a special case of matrix multiplication and simplifies the parameterizing of

the application, since the input matrix can only be a square NxN matrix and the output matrix will be of the same size. We have also used a straightforward implementation with no algorithmic optimization. In this implementation, squaring an NxN matrix requires N^3 multiplications and N^2(N-1) additions. The primary computational resource in this implementation is the hardware floating-point units of the FPGA. Figure 3.7 shows the block diagram of the device. It is essentially a single multiply-accumulate module controlled by a controller that reads the necessary elements from memory; that computes one multiplication at a time; accumulate the results to calculate the sum which constitutes one element of the result matrix; and finally writes the result to memory and goes to the next element. Arguably the bottleneck of the computation is the floating-point multlipication and accumulation. Therefore, the MAC module can be considered as the computational core of the device. In a sense, hardware acceleration of squared matrix can be perceived as delivering the performance of hardware floating-point units to the software. There is also a pipeline register on the multiplier output that can be configured to be used or bypassed. The host application for this implementation runs on a NIOS processor and the host-device communication is through a shared block RAM on Altera Avalon bus. This memory is used once as a single buffer and once as a double buffer and two memory-mapped registers are used for synchronization.

4.3 Results

The following results are obtained by running a host application that calls a large number of consecutive invocations and measures the achieved throughput in software. Each measurement gives us a single point on the diagram. The primary concern of the evaluation is to verify that these points correspond to our expections of the model. These results will be used in the next section for analysis and illustration of the FPGA-Roofline capabilities.

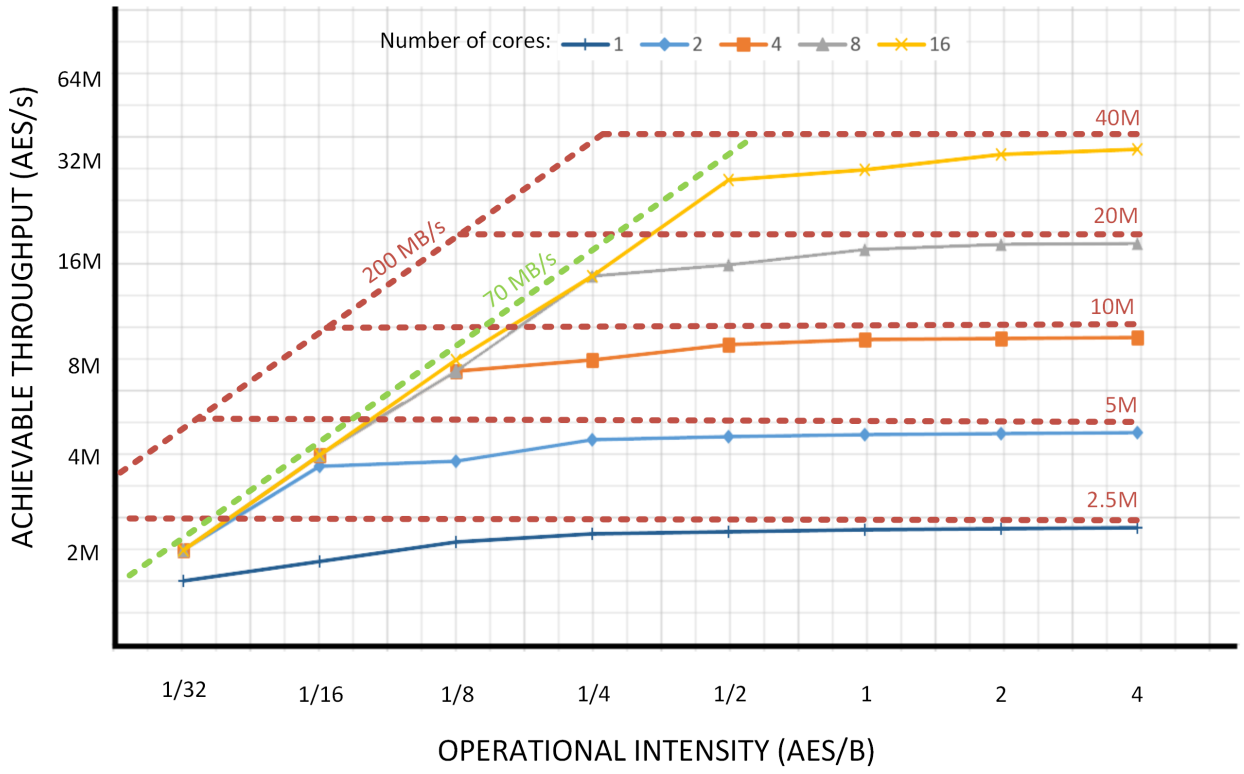


Figure 4.1: FPGA-Roofline model and the experiment results for the implemented AES hardware accelerator

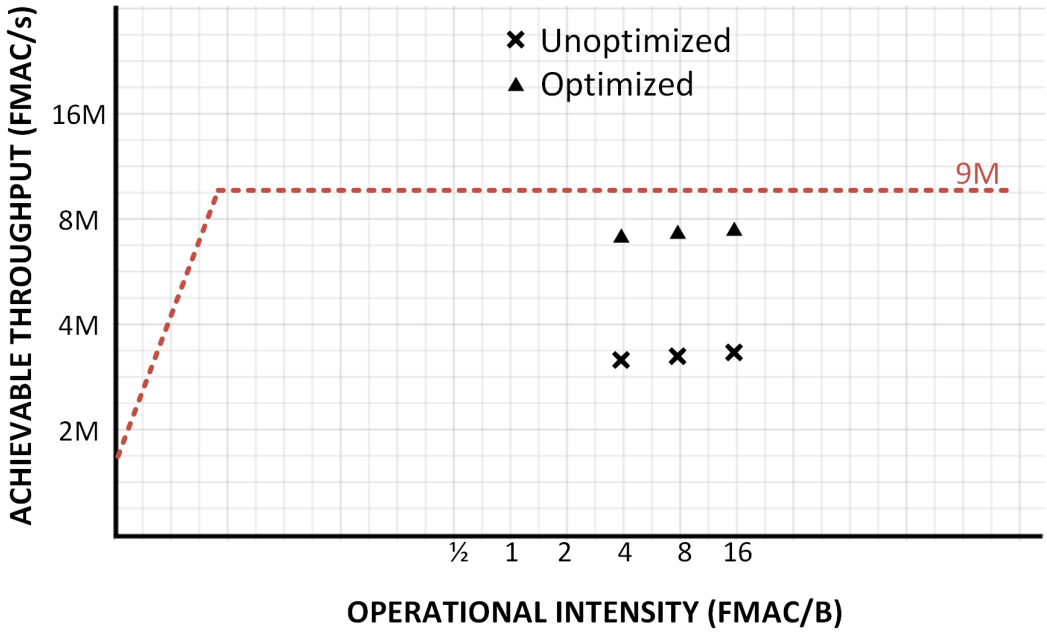


Figure 4.2: FPGA-Roofline model and the experiment results for the implemented squared matrix hardware accelerator. Floating-point multiply-accumulate is defined as the unit operation.

AES: Figure 4.1 shows the results for AES implementation. The red dashed lines are for the roofline model that is created based on system specifications before the implementation. The peak throughput of a single AES core in our implementation is 2.5M AES/s and the peak bandwidth of PCI Express is roughly 200 MB/s. These two specifications form the roofline. The slope of the diagonal part of the roofline is 200M and the height of the horizontal part is 2.5M multiplied by the number of active AES cores. The actual measured points are also marked on the diagram. Points corresponding to the same number of cores are joined together with a colored line. The first observation is that the achieved throughput in software is indeed below the bounds of the roofline in all cases. Further analysis and discussion will be provided in next section.

Matrix Squaring: Figure 4.2 shows the results for squared matrix implementation. The latency of the floating-point unit is 11 clock cycles that results in peak throughput of 4.5M FMAC/s at 50 MHz. FMAC stands for floating- point multiply-accumulate and is considered as the unit operation in this model. The peak bandwidth of Altera Avalon bus is 4 bytes per clock cycle which is equivalent of 200MB/s at 50MHz. For an NxN matrix, $2N^2$ bytes of data are transferred and N^3 FMAC operations are done per invocation. As a result, the operational intensity for squaring an NxN matrix is equal to N/2. The experiments are done for three different sizes of 8x8, 16x16 and 32x32 that respectively make for the operational intensities of 4, 8 and 16. There is an optimized version of the device that will be discussed in Section 5.5. The measured points are marked on the diagram along with the roofline. It can be observed that the achieved throughput is below the bounds of the roofline. Moreover, as one could predict by the roofline, the throughput for all matrix sizes is almost the same. Further analysis and discussion is provided in Chapter 5.

Chapter 5

Analysis and Insights

In this chapter we discuss several aspects of FPGA-Roofline with more details. Each section is devoted to an important discussion that provides extra insights into our model and how it can be improved and used efficiently in design and analysis of hardware accelerators.

5.1 Comparison of FPGA-Roofline and the Roofline Model

The original Roofline model explained in Section 2.2 is the basic inspiration of FPGA-Roofline. Although they share a principal concept that is based on concurrency as explained in Section 2.3, they have fundamental differences in methodology and application. The main difference of these models is rooted in the fundamental difference of general purpose computing and embedded computing as discussed in Section sec:hpc-vs-embedded. Our model aims to accelerate the process of hardware-software integration in an embedded system. Embedded systems are built around a specific application. Hence, FPGA-Roofline factors the application into modeling. That is why the unit of Y-axis in our model is OP/s and OP is application-dependent. On the contrary, the Roofline model uses FLOPS as a generic per-

formance unit in HPC applications. This reflects the general purpose nature of HPC point of view. Another difference is the architectural modeling of the system. We use a host-device model as explained in Section 3.1, while the Roofline model uses a CPU-Memory model. As a result, computation is being done on different components and the communication link has a different purpose. In overall, FPGA-Roofline adopts the modeling approach of the Roofline model —which was previously unknown to embedded system designers— and adjusts it in a way such that suits the needs of embedded system design.

5.2 Improving Precision and Reusability by Benchmarking the Link

In Figure 4.1, the link bandwidth of 200 MB/s, drawn by a red dashed line, is used to form the roofline. However, another diagonal line with 70 MB/s is drawn by a green dashed line that obviously has resulted in a more precise roofline. The first value is extracted from the specifications of PCI Express, while the second value is measured on DE2-i150 platform. We have benchmarked our implementation of PCI Express connectivity and the results showed that our configuration of PCI Express and Xillybus could reach up to 70 MB/s. Therefore, we can improve the precision of our model by using benchmarking results instead of the nominal performance in the specifications. The specifications of hardware modules usually provide their theoretical peak performance. These values are sometimes impractical to achieve in real implementations due to inevitable and sometimes predictable overheads. An ambitious number results in an ambitious model that overestimates the capability of the system.This problem can hurt the roofline model precision by misapproximating the peak bandwidth of the link. For example, the real bandwidth of a PCI Express implementation depends on many factors such as the hardware module, switch-level configurations, choice of design parameters, operating system, buffering scheme, etc. This complex set of parameters makes it practically impossible to predict the actual peak bandwidth. To have a more precise

model we need more realistic specifications of the system than can be achieved through benchmarking. A simple approach to benchmark a communication link is by profiling it while running a loop-back test. To have more useful benchmarking results, this can be done under different configurations of the link if applicable. The downside of benchmarking compared to using the specifications is the requirement of physical access to the platform and the benchmarking effort. Still, there is no need to implement the device logic in FPGA to make the model. Moreover, benchmarking is a one-time process and needs to be done once per platform. Hence, it can be done once by a platform expert and the results can be published to be reused by other developers for different applications. As an example, we have benchmarked the PCI Express connectivity in the platform used for the AES example. Figure 5.1 shows the results of a loop-back benchmark for different amounts of data. The loop-back time and the corresponding throughput are shown on the diagram. The step-by-step increasing fashion of the loop-back time is a known effect imposed by cache hierarchy. The operating system uses memory blocks as buffers to transmit or receive data over PCI Express. The choice of the block size along with the architecture cacheline size and cache block size may affect the details of this step-by-step trend. This diagram is an example of useful benchmarking result that can increase the precision of the model. Using this diagram, The developer can find out the efficient size for user space buffer size that is used for data transfers. We have done this in our AES example, and as it is reflected in Figure 4.1, our communicate-bound implementations are close to the 70 MB/s diagonal line which is the upper-bound of data transfer suggested by Figure 5.1.

5.3 Effective Parallelism

The main advantage of hardware compared to software is concurrency. Multiple computations can be done in parallel. Leveraging the parallelism is one of the usual motives for hardware acceleration in FPGA. A common issue with parallelization of the computation logic is that with more parallelization more bandwidth is needed for communication. Hence,

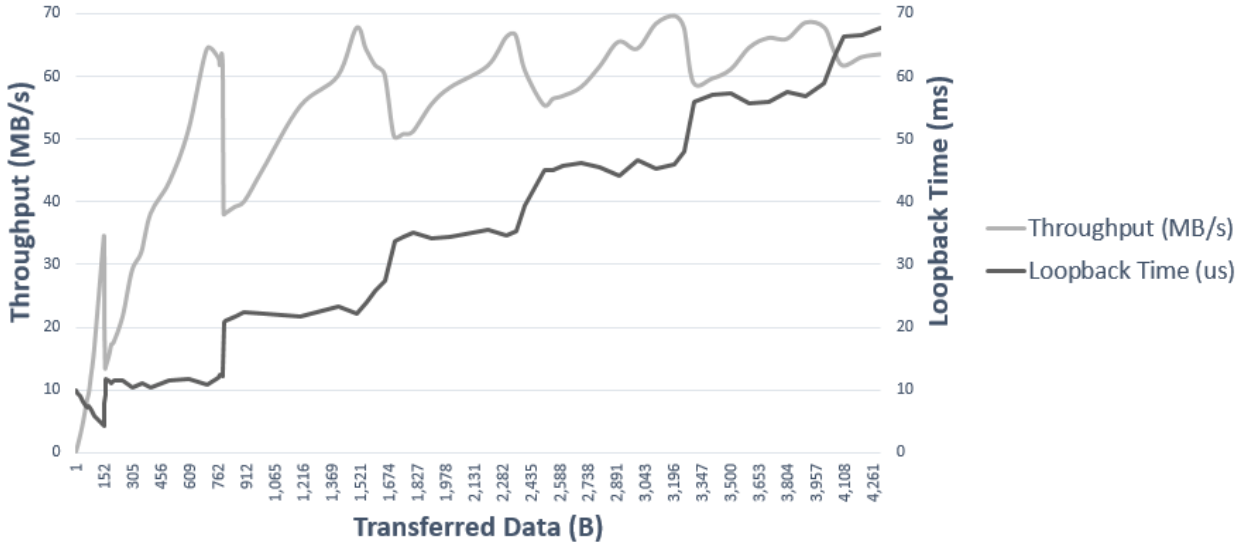


Figure 5.1: Loop-back benchmark results on our PCI Express connectivity

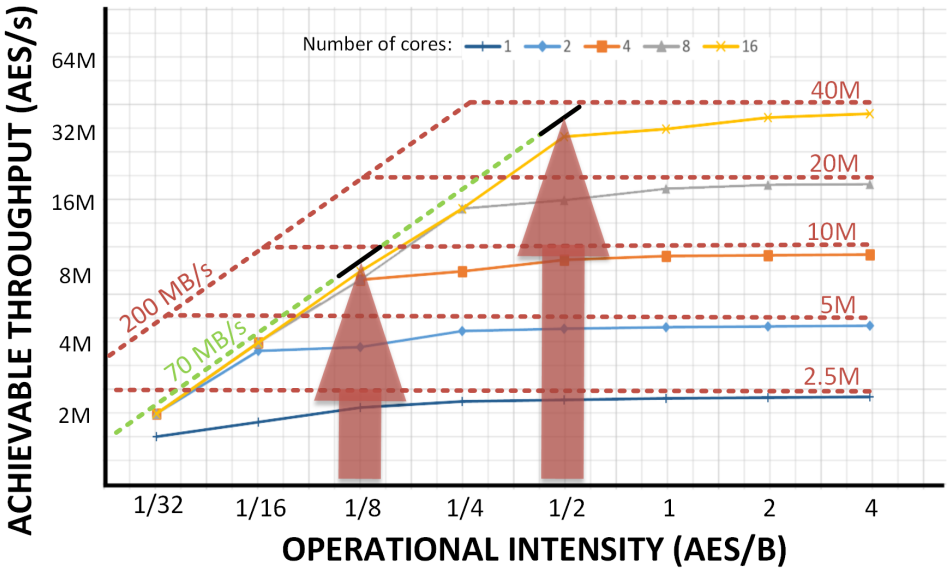


Figure 5.2: FPGA-Roofline insights for effective parallelization

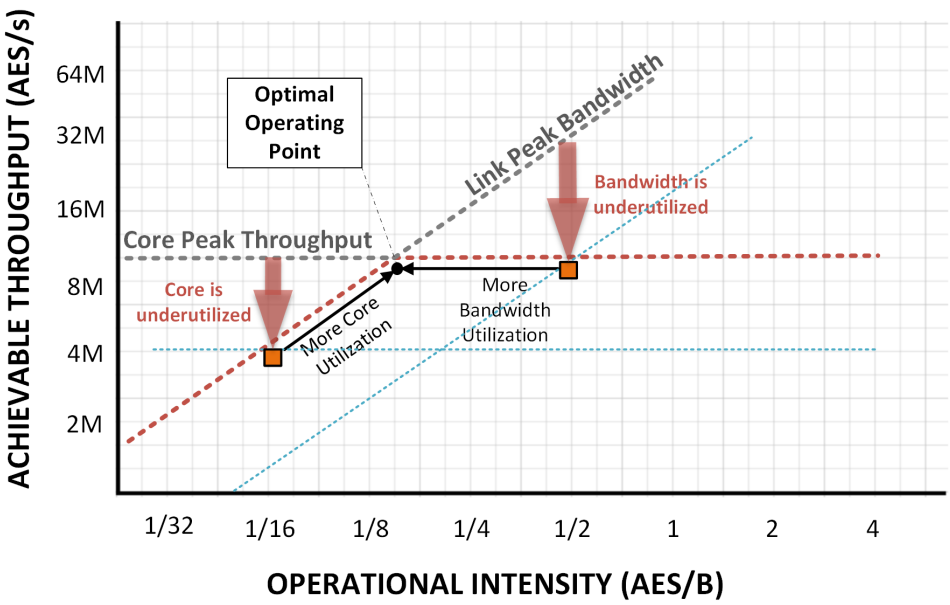


Figure 5.3: FPGA-Roofline insights for resource utilization

having a bigger FPGA simply does not mean more throughput in the system. FPGA-Roofline is insightful in showing the right amount of parallelization that can be effective in increasing the system throughput. Figure 5.2 shows an example of this in our AES implementations. With operational intensity of 1/8 the throughput is increased from around 2M AES/s for one core to around 4M AES/s for two cores and around 8M AES/s for four cores, but it stays at the same value for eight and sixteen cores. The reason is also easy to observe in the diagram: the achievable throughput for this operation intensity is bound by the diagonal roofline that corresponds to the peak link bandwidth of 70 MB/s. Therefore, it does not make sense to use more cores in parallel since they cannot contribute to the system throughput. Similarly, for the operational intensity of 1/2 using up to sixteen cores would benefit the system throughput. This analysis can be done prior to implementation to give a realistic expectation of parallelization benefit.

5.4 Resource Utilization Analysis

FPGA-Roofline can be also insightful to analyze the resource utilization of the system and potentially reduce the resource cost without hurting the performance. The ridge point of the roofline represents an optimal operating point of the system when the computation logic in the device and the link bandwidth are fully utilized, while the maximum throughput is also achieved. This implies that if our application has the same operational intensity of the ridge point, it is an optimal application in terms of potential utilization of the system resources. The further we move to the right horizontally, the less utilized is the link bandwidth. Similarly, moving to the left means less utilization of the device cores. This analysis is demonstrated in Figure 5.3. Two measurements of our AES implementation are shown on FPGA-Roofline as examples. One of them is bound by the core throughput and underutilizes the link bandwidth, while the other one is bound by the link bandwidth and underutilizes the core throughput. The red arrows show the underutilizations. This insight can be very helpful in deciding if our platform is suitable for the application. Moreover, it helps to adjust the system to get to its optimal operating point. For example if the link bandwidth is underutilized we can use a different configuration of the link bandwidth which can benefit us in another aspect such as power consumption; or we can simply use a cheaper hardware for communication if it is a design choice and save some production cost. If the device is being underutilized we can do adjustments like decreasing the clock frequency if possible or using a low-power mode that delivers lower performance. As a result, the throughput delivered to the software can remain the same while some resource cost is reduced. The dotted lines in Figure 5.3 show the less costly alternative resource/configuration that would put the system in an optimal operating point.

5.5 Optimization

FPGA-Roofline offers two candidates for the bottleneck of throughput: peak bandwidth of the link and peak throughput of the core. If an implementation does not achieve the throughput bounds indicated by the roofline, either the roofline is not formed precisely or the implementation has performance overheads. If the overhead is significant, optimization becomes necessary. FPGA-Roofline can offer insights for optimizing the system for maximum throughput and fill the gap between the performance of an inefficient implementation and its anticipated performance on the roofline.

Optimizing the link bandwidth: The key point for optimization is to optimize the bottleneck. Therefore, when throughput is bound by the link bandwidth (left side of the ridge point), only the optimizations that improve utilization of the link bandwidth will improve the system throughput. The technical details of the optimization depends on the hardware. It can be a change in configuration, software driver, operating system, etc. Experts can study the platform and provide the possible optimization techniques to developers. Each optimization results in a different peak bandwidth that can be reflected on the roofline as a ceiling. In Figure 4.1, the green line indicating 70 MB/s is a ceiling for our configuration of the link. If someone found a more efficient configuration with better performance, a new ceiling could be added to the roofline. Optimization ceilings give quick visual insights to developers about the options for optimization and their gain.

Optimizing the device throughput: Optimization may be more tricky when throughput is bound by the peak throughput of the core (right side of the ridge point). In this case, we cannot simply say we should improve the core throughput, since in Section 3.1 we have defined the core as the unalterable part of the hardware that provides the computing power. Thus, the core itself is a black box with a fixed latency. Tampering the core would mean changing its peak performance and changing the roofline itself. Looking back at Figure 3.3.b we can see that the ideal operation of the system is when the device is fully utilized and the core is always busy with no delay between the consecutive operations. In this case,

overhead is anything that delays the core between two consecutive operations. There are two latencies that can cause this delay: the latency of I/O data transfer, and the latency of synchronization. Both of these factor can be optimizable depending on the platform and the application. Sometimes the latency can be reduced, but most of the time there is a physical limit to the latency and the solution is to hide the latency by using buffers. In advanced platforms, the synchronization and buffering are done in underlying layers that can achieve a high level of latency hiding. For example, the synchronization for PCI Express is done completely in hardware and the capability of defining multiple channels with multiple Direct Memory Access (DMA) buffers assigned to multiple threads on the CPU ensures the feasibility of hiding latency to a great level. Our AES implementation could achieve the roofline bounds quite well by leveraging the capabilities of Xillybus and PCI Express. In less sophisticated platforms, communication issues should be handled manually and hiding latency can be more challenging. As Figure 4.2 suggests, the unoptimized version of the squared matrix implementation has a significant inefficiency. In this implementation, the device is responsible for reading from and writing to memory for communication and also make use of the core for computation. Hence, utilization of the core can vary based on the fashion in which these steps are taken by the device. In our basic implementation, the device does every step in order with no overlap. Memory read, multiplication, accumulation, and memory write operations have no pairwise overlaps and hence there is no hiding of the latency. The optimized version however, makes use of the pipeline register in the core, and loads the next two elements from memory for next multiplication while the accumulate process is being done for the current elements. By this proper use of pipelining, the device can divide the process steps and overlap them to hide latency of the memory and floating point units. Figure 4.2 shows the significant improvement of this implementation over the basic one.

5.6 Bandwidth vs. Latency

In Section 5.5 we discussed how FPGA-Roofline can be insightful for optimization. In summary, we said that in the case of being bound by the link bandwidth, platform-specific techniques for improving the bandwidth utilization must be used. In case of being bound by the core throughput, the latency must be reduced or hidden. The reality of hardware acceleration is that we ideally want to achieve the peak throughput of the core, so being bound by the core throughput is desirable compared to being bound by the link bandwidth. So it is more probable that we end up in the right side of the ridge point in the roofline, and it also continues to be increasingly probable in the newer platforms according to David A. Patterson [7] stating that "Latency lags bandwidth" by providing evidence that bandwidth is improving faster than latency. As a result, the slope of the diagonal part of the roofline increases causing the ridge point to shift to the left side, leaving more applications on the right side. Moreover, latency issues cause more inefficiency in the integration and make it harder for implementations to *hit* the roofline. Thus, it is imaginable that the most common challenge in a typical hardware accelerator integration is to deal with latency rather than bandwidth.

5.7 The Summary of FPGA-Roofline

To put everything together, we have summarized all the insights and visual assets that FPGA-Roofline can provide in one diagram. The more we know about the system specifications, its realistic performance bounds, and its optimization techniques, the more precise and insightful this model can get. As Figure 5.4 suggests, the following insights can be derived from FPGA-Roofline: Achievable peak throughput of the accelerated operation, bottleneck of the throughput, optimization opportunities, resource utilization opportunities, and effective parallelization factor. A key point about FPGA-Roofline is that it can be formed once per platform to be reused for different integrations and applications. A quick visual overview

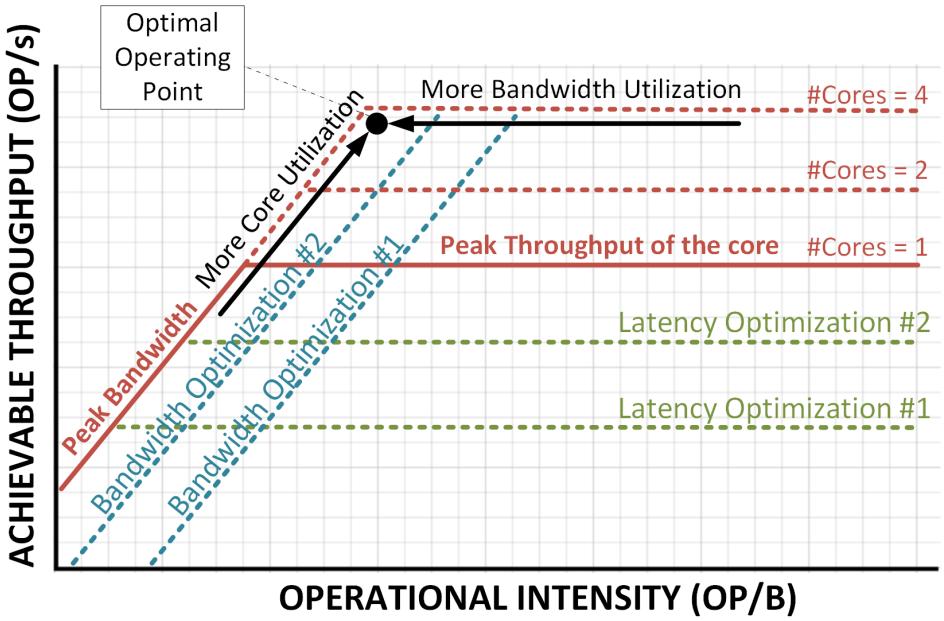


Figure 5.4: The summary of FPGA-Roofline

of the system gives a realistic and comprehensive understanding of its capabilities to developer and makes it easy to study and analyze the feasibility of accelerating an application in hardware. We believe that our model can be used as a common language to view any hardware accelerated system and a fast way to visually communicate the basic concepts of hardware acceleration without going into deep details of the implementation. Indeed, one could use FPGA-Roofline to explain, analyze or justify the different aspects of a hardware accelerator in a compact yet comprehensive fashion.

Bibliography

[1] Xillybus. [Online; accessed: 2014-11-20 at www.xillybus.com].

[2] Accelerating high-performance computing with fpgas. Altera Corporation, October 2007. [Online; accessed: 2014-11-20 at http://www.altera.com/literature/wp/wp-01029.pdf].

[3] J. Little. A proof for the queuing formula: L = λw. *Operations Research*, 9(3):383387, 1961.

[4] M. R. Meswani, L. Carrington, D. Unat, A. Snavely, S. Baden, and S. Poole. Modeling and predicting performance of high performance computing applications on hardware accelerators. *International Journal of High Performance Computing Applications*, 27(2):89–108, May 2013.

[5] A. M. Michael Barr. *Programming Embedded Systems: With C and GNU Development Tools*. O'Reilly, Sebastopol, California, 2007.

[6] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[7] D. A. Patterson. Latency lags bandwith. *Communications of the ACM*, 47(10):71–75, October 2004.

[8] T. Ramdas and G. Egan. A survey of fpgas for acceleration of high performance computing and their application to computational molecular biology. In *TENCON 2005 2005 IEEE Region 10*, pages 1–6, Nov 2005.

[9] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *Design and Test of Computers, IEEE*, 27(2):23–33, December 2006.

[10] S. Sarkar, T. Majumder, A. Kalyanaraman, and P. Pande. Hardware accelerators for biocomputing: A survey. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 3789–3792, May 2010.

[11] P. Schaumont and I. Verbauwhede. The exponential impact of creativity in computer engineering education. In *Microelectronic Systems Education (MSE)*, pages 17–20. IEEE, June 2013.

[12] R. Usselmann. Advanced encryption standard/rijndael ip core. *http://www.asic.ws/*, 2004.

[13] S. Williams, A. Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, April 2009.