Enabling Real-Time Mobile Cloud Computing through Emerging Technologies

Tolga Soyata
University of Rochester, USA

A volume in the Advances in Wireless Technologies and Telecommunication (AWTT) Book Series

Information Science REFERENCE
An Imprint of IGI Global

Managing Director: Lindsay Johnston
Managing Editor: Austin DeMarco
Director of Intellectual Property & Contracts: Jan Travers
Acquisitions Editor: Kayla Wolfe
Production Editor: Christina Henning
Development Editor: Brandon Carbaugh
Cover Design: Jason Mull

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA, USA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Enabling real-time mobile cloud computing through emerging technologies / Tolga Soyata, editor.
pages cm
Includes bibliographical references and index.
ISBN 978-1-4666-8662-5 (hc) -- ISBN 978-1-4666-8663-2 (eISBN) 1. Cloud computing. 2. Mobile computing. I. Soyata, Tolga, 1967-
QA76.585.E55 2015
004.67'82--dc23

2015015533

This book is published in the IGI Global book series Advances in Wireless Technologies and Telecommunication (AWTT) (ISSN: 2327-3305; eISSN: 2327-3313)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 6
Selling FLOPs: Telecom Service Providers Can Rent a Cloudlet via Acceleration as a Service (AXaaS)

Nathaniel Powers
University of Rochester, USA

Tolga Soyata
University of Rochester, USA

ABSTRACT

To meet the user demand for an ever-increasing mobile-cloud computing performance for resource-intensive mobile applications, we propose a new service architecture called Acceleration as a Service (AXaaS). We formulate AXaaS based on the observation that most resource-intensive applications, such as real-time face-recognition and augmented reality, have similar resource-demand characteristics: a vast majority of the program execution time is spent on a limited set of library calls, such as Generalized Matrix-Multiply operations (GEMM), or FFT. Our AXaaS model suggests accelerating only these operations by the Telecom Service Providers (TSP). We envision the TSP offering this service through a monthly computational service charge, much like their existing monthly bandwidth charge. We demonstrate the technological and business feasibility of AXaaS on a proof-of-concept real-time face recognition application. We elaborate on the consumer, developer, and the TSP view of this model. Our results confirm AXaaS as a novel and viable business model.

INTRODUCTION

Consumer use of "smart" devices is rapidly increasing due to affordability and increasing wide area network (WAN) performance (Emarketer, 2014). As the capabilities of smart phones expand parallel to the improvement in the WAN performance, so do consumers' expectations for resource-intensive mobile applications. However, mobile devices are ill-suited to execute most these applications due to their hardware limitations. Computational offloading offers a way to augment mobile computation power,

DOI: 10.4018/978-1-4666-8662-5.ch006

but it introduces a communication latency, potentially weakening or negating its advantages. Mobile-cloud computing expands the utility of mobile devices by enhancing the apparent performance of their applications. Offloading data storage and processing from mobile devices to the cloud allows a mobile device to appear more powerful than it really is (Chen, Ogata, & Horikawa, 2012; Soyata, Ba, Heinzelman, Kwon, & Shi, 2013; Cuervo, et al., 2010; Mei, Shimek, Wang, Chandra, & Weissman, 2011; Chun, Ihm, Maniatis, Naik, & Patti, 2011).

Although intensive processes are not actually being handled by the device, the requisite simplicity of the user interface effectively renders the offloading routines as transparent, leading to increasing expectations for performance by consumers, who are by now, quite used to experiencing steady improvements as the norm. An emerging class of mobile applications such as real-time face recognition, linguistic processing/translation and augmented reality (Keller, 2011; Kovachev, D., Cao, Y., & Klamma, R., 2013; Soyata T., Muraleedharan, Funai, Kwon, & Heinzelman, 2012) depend on computationally intensive methods, far surpassing the capabilities of mobile devices. Furthermore, performance improvement potential of these applications due to offloading is limited because of the added communication delays for transporting the code and data during offloading. To enable these applications, solutions based on an intermediary cloudlet device have been proposed to accelerate the computation partially at the source before it reaches the cloud (Kwon, et al., 2014; Verbelen, Simoens, DeTurck, & Dhoedt, 2012; Soyata T., et al., 2012; Satyanarayanan, Bahl, Caceres, & Davies, 2009). These techniques rely on an expensive cloudlet that has substantial computational power. Even if a mobile user makes an investment in such a cloudlet, continuous upgrades will be necessary to keep up with the increasing computational demand from evolving applications.

The commonly employed mobile-cloud model involves connecting to the cloud on a per-user basis. There is no service archetype in place which offers a standardized acceleration of computation for a large number of users. We introduce the AXaaS service model which can potentially connect millions of mobile device users to superior computational resources through the low latency links provided by Telecom Service Providers (TSPs). In our model, TSPs can be thought of as renting a cloudlet to their subscribers. This cloudlet can be orders-of-magnitude faster than one that their users can purchase. Since our target applications demand the use of this cloudlet in a burst form, the accumulated usage is actually very low. This provides an opportunity for the TSPs to aggregate these burst requests from millions of users and service them in powerful datacenters, enabling the aforementioned new class of mobile applications that are currently impossible or impractical to develop. We propose Telecom Service Providers (TSPs) as the most logical provider of AXaaS, due to their existing relationship with their users. By charging a monthly fee (e.g., TFLOPs/month), TSPs may use this revenue to offset the cost of renting computational resources from cloud operators (Amazon EC2, 2013; Azure, 2014). This model absolves the TSP of the burden of building and maintaining proprietary data centers. A variation of this model may also be realized by TSPs that do have existing cloud infrastructures, such as Verizon's Terremark (Verizon-Terremark). The central impetus is enabling universal access to superior computational resources in order to facilitate innovations in the development of mobile applications.

The rest of this chapter is organized as follows: We provide background on existing aaS models and motivate our proposed new aaS offering called Acceleration as a Service (AXaaS). Next, we detail a real-time face recognition application and identify its components that would benefit from acceleration. Based on this brief study, we make a distinctive definition of *acceleration* (which we stylize AX) and consider the types of constituting computations that would be considered AX. An example of the AXaaS model is examined along with an analysis of TSP costs and return on investment (ROI) for the TSP, followed

by the implications of the AXaaS offering to end-users. A developer's perspective is discussed in detail, focusing on the development of applications under the AXaaS environment. We explore performance evaluations and make our conclusions at the end of our chapter.

BACKGROUND AND MOTIVATION

"As a service" (aaS) offerings provide convenient, configurable solutions to computational problems by allocating resources over the internet (Dsouza, Kabbedjik, Seo, Jansen, & Brinkkemper, 2012; Costa, Migliavacca, Pietzuch, & Wolf, 2012; Sandikkaya & Harmanci, 2012). To run an exciting new breed of resource-intensive mobile applications such as Real-Time Face Recognition (Powers, Alling, Gyampoh-Vidogah, & Soyata, 2014), Augmented Reality, Real-time Language Translation (Google-Translate; MS-Translator), and Surveillance, it is possible for a user to rent cloud instances. However, the steps required for such a process will discourage even the most willing user. In this section, we will introduce a framework which will enable these applications with nearly zero burden to the user.

Distinguishing AXaaS from Other aaS Services

Our profiling of the aforementioned resource-intensive mobile applications reveal their continuous access to a limited set of highly parallelizable APIs such as, Generalized Matrix-Matrix Multiplication (GEMM) and Fast Fourier Transform (FFT). These applications spend a majority of their time executing these APIs, which can be accelerated through hardware accelerators (e.g., GPUs (Nvidia CUBLAS; Nvidia CUFFT)), thereby significantly improving their performance. This observation lets us conclude that, what our targeted applications need is burst access to extremely high-intensity computation, rather than continuous access to steady, generalized computation. We observe a lack of cloud service offerings for providing generalized burst raw computation to improve the performance of these mobile applications. In this vein, we propose a new service model which shifts its focus from the application to the fundamental APIs, which are very common to many resource-intensive applications. The impact of this shift we are proposing is significant. The only parameter that the service provider has to focus on is providing these APIs in an extremely performance-improved fashion (i.e., "AX amenable"). In other words, while providing the application as a service (i.e., the SaaS model) requires expertise of this software, providing Acceleration as a Service (AXaaS) requires only expertise in optimally trafficking the data-to-be-accelerated over the internet and providing acceleration for these very common APIs. In analyzing the AXaaS model's business viability, we will show that, the acceleration that is offered as a service can dramatically improve user experience, motivating the users to pay for such a new service.

Motivation for Acceleration as a Service

To illustrate the impact of acceleration, we will assume that, a user is running a mobile-based Face Recognition (FR) application on a state-of-the-art mobile phone or tablet over a 4G cellular network. We are particularly interested in performance differences owed to computational offloading. Our assumptions on application parameters are: 1) An image size of 145 KB, 2) result data size of 8 KB for the set of recognized faces, 3) a database size of 5000 faces to be recognized from, 4) Apple iPhone 5S based on the A7 Cyclone with PowerVR G6430 GPU (iPhone-5s) (3.24 double GFLOP/s compute capability),

Figure 1. Example picture frame with 14 faces. File size=145KB.

and Nvidia SHIELD tablet, based on the Tegra K1 (Nvidia-Shield, 2014) (15.2 double GFLOP/s) as the mobile platforms running this application, 5) a cloud compute capability of 500 TFLOP/s and 6) WAN speed of 10Mbps (4G Network).

Scenario 1: Assuming that the entirety of the FR application and its data are hosted and executed on the iPhone, our experiments allow us to estimate a single face to be processed and recognized by the iPhone in approximately 4.6 seconds, and by the Nvidia SHIELD in approximately 0.98 seconds. The technical details of the application will be provided in the next section.

Scenario 2: The tablet uploads the captured 145KB image containing 14 faces shown in Figure 1 through a 4G network to some acceleration instance. This instance would rapidly accelerate FR functions and return the result to the mobile back over the 4G. While this approach introduces a communication delay of 249.2 ms, the resultant query time would be only 249.59 ms. Merely 390 μs of rapid computation in the cloud is required to produce a complete set of results.

Scenario 3: While the first two scenarios are based on widely available realistic cell phone network speeds today, if we assume a more futuristic 1Gbps network speed (e.g., 5G, which should be offered within a few years (3GPP-TS23.401, 2008)), the response time to recognize the 80 faces in a single frame shown in Figure 2 would be reduced to 22.84 ms total (20.44 ms is the communication cost).

While a broad array of applications can benefit from AX, the general characteristic of candidate applications for AXaaS involve a significant amount of computation for a small amount of data. For example, in a candidate remote health monitoring application (Kocabas & Soyata, 2014; Kocabas, et al., 2013; Page, Kocabas, Ames, Venkitasubramaniam, & Soyata, 2014; Page, Kocabas, Soyata, Aktas, & Couderc, 2014), Fully Homomorphic Encryption (Gentry, 2009) is used to achieve health data privacy during data transfers as well as computation in the cloud. While FHE-encrypting the data at the acquisition source, such as the patient's house, would imply a high communications overhead, converting the data at the source from AES to FHE via existing algorithms (Gentry, Halevi, & Smart, 2012) completely changes this ratio, thereby making this application AX-amenable. In such a case, the data is only converted to AES, which is a very computationally-light process (NIST:FIPS-197, 2001) by using today's modern mobile devices (iPhone-5s), however, the extremely compute-intensive FHE-based computations can

Figure 2. Example picture frame with 80 faces. File size=1.29MB.

be performed by purchasing AXaaS accelerations. Note that, this conversion also involves a substantial amount of matrix operations (Gentry, Halevi, & Smart, 2012), which we will prove to be true for almost any AXaaS candidate application.

Quantifying the User Experience: The γ

Our key question to formulate a model for AXaaS is: how do these results translate to user experience? Clearly, users will only be willing to pay for services that either enhance their business, daily lives, or for entertainment. Our careful observation of the FR algorithm shows that, it is actually fairly simple to quantify the user experience. Let us define the following metrics:

$$\gamma = K * \lambda \tag{1}$$

where K is the average faces per frame (frame density) and λ is the average frames per second (temporal density). It is clear that, γ (*total faces recognized per second*) is highly related to user experience, but what range of γ values correspond to what type of user experiences ?

In the case of our Scenario 1, when a smart phone or tablet solely recognizes faces, we calculate γ=0.22 and γ=1.02, respectively. In the case of our Scenario 2 and Scenario 3, significantly improved γ≈50 and γ≈3500 are achieved. These simple examples show that, while an advanced mobile device that is available today struggles to recognize even a single face within a second, by augmenting the FR

application using AXaaS, the perceived value of the application can be drastically improved, thereby prompting the user to pay for such a service. While most users will happily call γ=10 a *Real-Time Face Recognition*, γ=10-100 will allow the application to be significantly more versatile, allowing the user to run it on highly populated scenes. Finally, γ=100-1000 and beyond will allow *Real-Time Surveillance* over very populated scenes, such as airports and public speeches of government officials.

THE FACE RECOGNITION ALGORITHM

In order to obtain a finer appreciation of the potential benefits of acceleration, we choose Real-Time Mobile-Cloud Face Recognition as our proof-of-concept application, which possesses many representative characteristics of applications that are amenable to AXaaS-based acceleration. In this section, we will study the computational aspects of our Face Recognition (FR) application in detail. The FR algorithm can be divided into three distinct phases: Face Detection, which separates and extracts the actual faces from the source frame, Projection, which converts each detected face into a set of coefficients, and Search, which compares the projection of the detected face with projections of images stored in a database and returns a result with the highest degree of similarity (Alling, Powers, & Soyata, 2015). In this chapter, we will detail each component of the overall face recognition process and will derive its computational requirements.

Detection

To execute detection we have employed the Viola-Jones object detection framework (Viola & Jones, 2001; Viola & Jones, 2001), which provides competitive real-time detection rates. This portion of the algorithm is very well studied (Yang, Kriegman, & Ahuja, 2002) and hardware-based acceleration techniques have been proposed (e.g., FPGA-based acceleration (Cho, Mirzae, Oberg, & Kastner, 2009)). Most modern smart phones have hardware accelerators for this very common function. Today's operating systems, such as Android 4.x, Ice Cream Sandwich or Jelly Bean (Andoid-API) have built-in API functions to take advantage of this hardware acceleration, albeit at a limited capacity of up to a maximum of two faces per frame. We have implemented face detection via GPUs using a frontal face cascade classifier using the open-source computer vision library OpenCV (Bradski & Kaehler, 2008) and Nvidia's CUDA (Nvidia CUDA) programming language. We find that the frame-face density (the K metric introduced in Equation) has a minimal impact on the detection rate for each frame, Table 1 shows detection times for different GPU architectures.

Table 1. Face Detection times for different devices (800×480 frame), with extrapolated estimates marked with (*).

Device	Peak GFLOPS/s	Detection Time (ms)
GTX 480	1345	58
GTX 760	2258	46
GTX 780	3977	35
TESLA K40*	4920	28
GTX Titan Z*	8122	19

Face Detection is highly parallelizable. Typically, a complete detection routine on a single frame involves the application of dozens of classifiers, leading to hundreds of millions of operations. Although we do consider detection to be a process that can significantly benefit from acceleration per se, its implementation outside of mobile devices with a detection capability might be to support extended application functions such as performing FR on scenes with high frame-face density (e.g., K≫5).

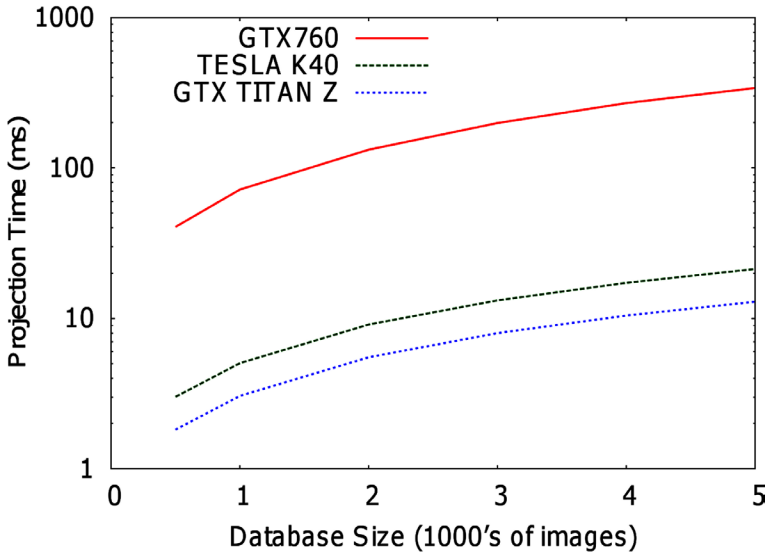
Projection

Projection has shown to be the most computation-centric component in our application. The Eigenfaces method (Turk & Pentland, 1991) in the OpenCV FaceRecognizer API (OpenCV-FaceRecognizer) calls for two nearly identical but separable components which are necessary for FR. The first is an off-line initialization which trains a multi-dimensional Eigenspace from database images. Using Principle Component Analysis (PCA) (Kim, Jung, & Kim, 2002), a set of near-orthogonal basis functions called eigenvectors are formed which allow each face in the database to be represented as a unique linear combination thereof. This allows for a significant reduction in data required to represent each face. Finally, a projection vector is generated for each image in the database and stored in a database file, along with the Eigenvectors. Application run-time requires the entirety of the database to be loaded into host memory, the local RAM of the platform running the application.

The second component occurs during FR run-time. The projection of the test face is calculated as a product of the test face (expressed as an array of floating point numbers) and the matrix of eigenvectors formed during the PCA. Since the number of eigenvectors is proportional to the number of images in the training database (equivalent in our case), it clearly follows that the number of operations (FLOPs) necessary to carry out the multiplication will increase with database size, which can be observed in Figure 3.

OpenCV implements matrix multiplication Level-3 BLAS (Basic Linear Algebra Subroutines) GEMM functions (Nvidia CUBLAS). Each call to GEMM involves uploading the test face ($8*n_{pixels}$ bytes in size) and the eigenvector array ($8*n_{pixels}*n_{components}$ bytes) to device memory (i.e., GPU memory).

Figure 3. DB-size dependent Projection times for different architectures.



A multi-threaded kernel is then launched which computes the multiplication. Kernel execution of the matrix multiplication consists of billions (i.e., Giga) of both integer and double-precision floating-point operations (denoted as GIOP and GFLOP, respectively).

Search

The final step compares each test face projection to the collection of projections stored in the database compiled during initialization. The prediction result corresponds to the projection in the database with the minimum Euclidean distance to that of the test face. The process time of our routine increases with the square of images in the database since the number of operations scales with the product of image count and number of components (dimensions) of the Eigenspace. The largest database we have tested consisted of 5000 images, each with a resolution of 180×180 pixels, and search times are negligible (<1 ms), since this operation is not compute-intensive.

COMPUTATIONAL ACCELERATION (AX)

In this section, we clearly define the difference between generalized computation vs. computational acceleration. By acceleration (stylized AX), we literally mean a computational impulse; where an extreme number of operations are carried out within an extremely small time interval. Since our target applications spend a vast majority of their execution time in a small set of generalized API calls, such as the generalized Matrix Multiplications (GEMM) as we discussed in the previous section, executing these GEMM calls 100's or 1000's of times faster (i.e., *accelerating* them) will have a dramatic impact on the user's application experience. An example application that may benefit from acceleration is real-time machine translation (MS-Translator; Google-Translate) in which speech is captured by a mobile device, translated into another language and either displayed as text or dictated to the user as audio. Computations used in this application are based on Hidden Markov Models, which rely heavily on processing Discrete Fourier Transforms and can be explicitly expressed as matrix multiplications, or computed directly using high-level libraries such as CUFFT (Nvidia CUFFT), CUBLAS (Nvidia CUBLAS), or CUSPARSE (Nvidia CUSPARSE).

Accelerating Face Recognition

While many other compute-intensive applications can be formulated as candidates for AX, without loss of generality, we will specifically focus on real-time face recognition (FR) in this section. Not all processes that require computation would necessarily benefit from acceleration, though we have shown with our FR application that Projection particularly would, due to the vast number of calls to the GEMM subroutines. This is a highly generalized operation that may be used by an incredibly broad array of applications. There exist other operations at the same abstraction layer which could be used similarly, such as Fast Fourier Transforms (FFTs), sorting and sparse matrix operations. Our key idea in this chapter is to accelerate only such generalized common functions, rather than the application itself, thereby eliminating the necessity to rewrite acceleration routines for every new application.

In the previous section, we introduced a metric to express the user-apparent performance of our FR application (i.e., user experience). γ is the total faces recognized per second, including every frame within that second. We also identified communication and computation as the pivotal deterministic factors for achieving values for γ large enough for the application to be considered useful to users. Providing acceleration (AX) for FR would undoubtedly require the most computationally intensive cloud resources, as well as low-latency communication pipelines between the user and the cloud. Each of these factors are inherently finite in their own regard, thus do present limitations to the amount of acceleration that can be realized. By focusing AX on projection and the underlying GEMM method calls, we will see that application utility lies within the span of possibility. Equation below is an expansion of Equation which clarifies the impact of related parameters on observable performance

$$\gamma^{-1} = \frac{DATA_{up}}{\mathrm{K} * BW_{up}} + \frac{DATA_{down}}{BW_{down}} + \frac{GFLOP_{func}}{GFLOP\,/\,s} + \frac{RTT}{\mathrm{K}} \tag{2}$$

where K represents the frame-face density (in faces per frame), *RTT* (round trip time) represents the network latency, $DATA_{up}$ and $DATA_{down}$ are the uploaded data and downloaded results, along with the bandwidths of the uplink and downlink (BW_{up} and BW_{down}). $GFLOP_{func}$ and *GFLOP/s* represent the total number of GFLOPS required to execute the required functions and the computational speed (in GFLOP per second), respectively.

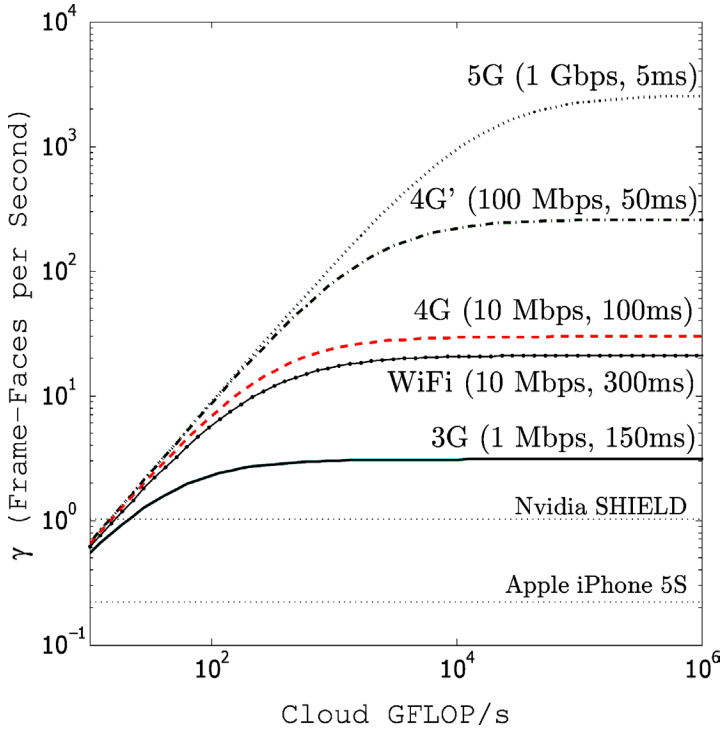
To maximize γ, we seek the maximization of each term's denominator (upload/download network transfer rates and computation rate). These are the most obvious. Notice that, by increasing the frame-face density K, we can affect an increase in γ by decreasing the ratio of uploaded data to the complexity of summary computation. Maximization of γ may also be sought by minimizing the numerators in each term (upload/download data size as well as contributions from *RTT*). $GFLOP_{func}$ represents the computational cost of a subroutine, for our case, detection, projection, and search, which might be reduced by increasing algorithmic efficiency. Figure 4 shows achievable γ as a function of cloud compute capability, network data rates/latencies as well as frame-face density. Data sizes are as specified previously.

Limitations of AX

The most obvious impediments to real-time performance are the bandwidth limitations and latencies associated with the transport layer. Considerations must be given to radio delays (i.e., 3G/4G/5G), IP Back-haul Transport latencies within the mobile service architecture and the connection between serving gateways (i.e., GGSN/PDNGW) and AX domains. One variant of the AX model places cloud domains in the internet by which they are accessed through a multi-hop connection. While this allows for universal access, it introduces the same IP delays associated with the classical mobile-cloud model (Wang, Liu, & Soyata, 2014; Soyata T., Muraleedharan, Funai, Kwon, & Heinzelman, 2012).

Another possible variant might imagine AX domains as having a direct connection to serving gateways, similar to IMS service provisioning (3GPP-TS23.228). While the 4G LTE Advanced standard claims to offer mobile peak rates up to 100 Mbps, most TSP migrations to the network technology in the U.S. are just beginning. However, we may reasonably assume a gradual increase in network performance and QoS expectations as they evolve.

Figure 4. Relationship between ɣ and cloud compute capability on various network generations, for FR application using a 5000-image database. The image in Figure 1 is assumed for WiFi and Figure 2 for all other networks. Network metrics are provided in (BW, RTT), e.g., 100 Mbps bandwidth and 50 ms Round Trip Time.



What Figure 4 illustrates clearly is that, 3G network speeds offer little to no acceleration potential for K=1 (one face per frame) in comparison to local execution on a mobile device. Single-face FR requests processed over 3G on a cloud instance even with a PFLOP/s capability barely approach γ=0.5, far from the baseline performance of the Nvidia SHIELD tablet. Each generation of network standards shows dramatic positive contributions to performance for any K, given a reasonably powerful cloud (10–100 TFLOP/s). Furthermore, each subsequent generation approaches a performance limit at much higher cloud compute capability, allowing for future exploitation of advancing computational resources.

Executing and Aggregating AX Requests

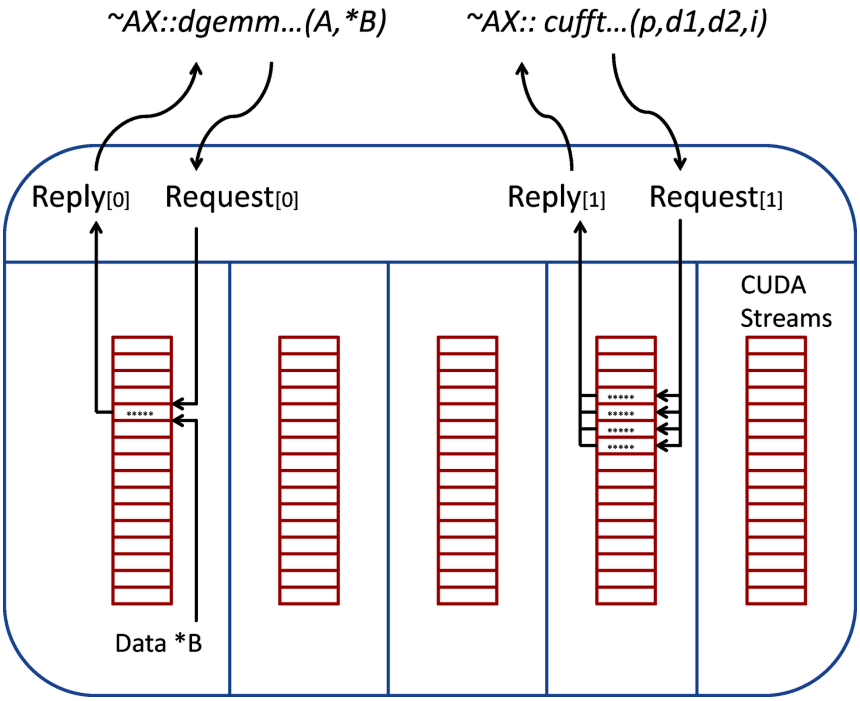
We consider a hypothetical cloud architecture of tightly coupled instances whose sole purpose is to execute AX Requests that are coming from *AX-aware* mobile application code. The nature of acceleration requires these computations to occur rapidly and relatively sparsely. We envision a large majority of these requests to execute on GPUs, as they can be utilized in a highly parallel fashion and offer competitive computational capacity for complex processes. Remote access to GPU clusters can be provided by routing traffic from TSP packet data network gateways (PDN-GW) over the internet to AX servers in the cloud. The Head Node in the cluster would be responsible for running the host-side AX requests and delegating requests to devices within the cluster architecture. NVIDIA GPUs are structured in

streams, which contain the processes of memory transfers and kernel launches. These streams can be run concurrently in Fermi and Kepler-based GPUs. AX would require the exploitation of this capability to maximize the arithmetic intensity of subroutines and serve multiple users' requests without delays related to process queuing.

The traversal of two AX requests through an example cluster are shown in Figure 5. We see the receipt of a valid AX request for GEMM at the Head Node, including the ID of the requesting device, the function to be performed, and the data on which the function is to be performed (A). Also included is an indication of what database information is required for the computation (*B, which for our purposes represents the FR eigenvector). The function is called into an open CUDA stream on a device, executed and returned to the user (indicated as "Reply" in Figure 5). Also present at the Head Node is a request for an FFT subroutine (denoted as "cufft" (Nvidia CUFFT)) along with all the requisite data. Supposing a higher Computational Quality-of-Service (CQoS) for this device ID, the function is spread over several streams, the graphic of which is meant more to reinforce the concept of CQoS, not the particulars of implementation with respect to the function type.

Multiple AX requests could be aggregated and spread across available resources intelligently by the AX execution *run-time* to ensure minimal response times and fairness for each user. Pushing a request forward to a cloud instance with a device whose streams are already saturated would result in undesired delays. By adding machine instances to the AX pool, or increasing the efficiency of device utilization, the potential for reliable acceleration for any number of subscribers increases. Since we envision a broad range of applications working simultaneously, we grant that resource awareness and allocation is critical in preserving the integrity and performance potential of the service. It may be possible to dedicate portions of the cloud hardware to handle specific data requests (e.g., dedicated GEMM instances or

Figure 5. Example structure of routed AX Requests and Replies.



dedicated FFT instances). Alternatively, hardware accelerators, instead of computers, may be used to accelerate commonly used AX requests (e.g., FPGA-based (Cho, Mirzae, Oberg, & Kastner, 2009)). As the variety of applications grows, we envision the developers to tailor their code to the accelerated APIs to provide higher-performance applications.

AXaaS: ACCELERATION AS A SERVICE

Our AXaaS model formulates the computational acceleration (AX) for mobile applications (detailed in the previous section) as a billable service (aaS) by the TSP. AXaaS proposes to offer incentives to every participating party: I) TSP, II) consumer, and III) the developer, but begins with a sensitivity to the demands of consumers. Mobile device users simply want better applications and better performance. Developers need an architecture in which these applications and levels of performance can be realized. Cloud operators possess the computational resources to make this happen, however a unified structure does not exist among these entities with a focus on computation. Although this type of support can be implemented on an individual basis, we envision universal availability to enable the widespread adoption of the AXaaS model.

We conceptualize the TSP as the central component necessary for bringing AX to the mainstream. By renting cloud resources on behalf of customers, computational acceleration can be made available to millions of existing TSP customers through a simple augmentation of a monthly service agreement. As developers begin to understand how far the performance boundaries can be pushed, they will flood the market with applications and programming interface extensions and optimizations designed to take advantage of the AX potential. Spurring competition among developers to create and release the "next big thing" would ensure no shortage of options for the consumer. The increasing diversity and utility of applications will attract customers to AXaaS-enabled applications, thereby leading to service subscriptions.

The status of AXaaS today is no different than the introduction of the monthly data-throughput plans, which did not see widespread adoption until the emergence of exciting services that absolutely required more throughput. Movie or music downloads, live-streamed football, baseball, soccer games etc. made these plans a "must" for many users. Such wide adoption prompted TSPs to offer ever-increasing data-throughput levels in their monthly plans to compete with each other. We see the potential of AXaaS being very similar to these services, with the exception of the service focus: *computation*. Existing services only focus on shuttling the data in and out of the mobile device, since the required computation can easily be achieved by the mobile device (e.g., streamed movies). With AXaaS, an additional major constraint is placed on the application: *computation*, which forces portions of the application execution to take place outside the mobile device. AXaaS intelligently decouples two portions of computation: I) general computation, which can easily be achieved on the mobile device, and II) accelerable computation, which is much less data intensive, but substantially more compute-intensive. The AXaaS model is based on the observation that only this second portion needs acceleration and can be obtained from the TSP *as a Service*, while the first part can comfortably take place on the mobile device.

An illustration of the relationship between the consumer, TSP, and cloud operator is given in Figure 6. We see the TSP as a customer to the cloud operator, and as a service broker to its own customers (i.e., the consumer). This model is desirable since the TSP already possesses a vast customer base. It is relatively easy for the TSP to increase mobile device users' awareness of new products and services. Additionally, the TSP already possesses a robust communication network which reaches millions of mobile device

users. There already exist cases where 4G is faster than WiFi (Huang, et al., 2012) and with the growth of LTE infrastructure, we expect this disparity to grow. By containing the majority of IP traffic on the TSP network, users are afforded greater connectivity, mobility and speed in comparison to a traditional multi-hop internet connection. Lastly, the TSP has the means to procure the amount of cloud resources necessary to service a large number of users.

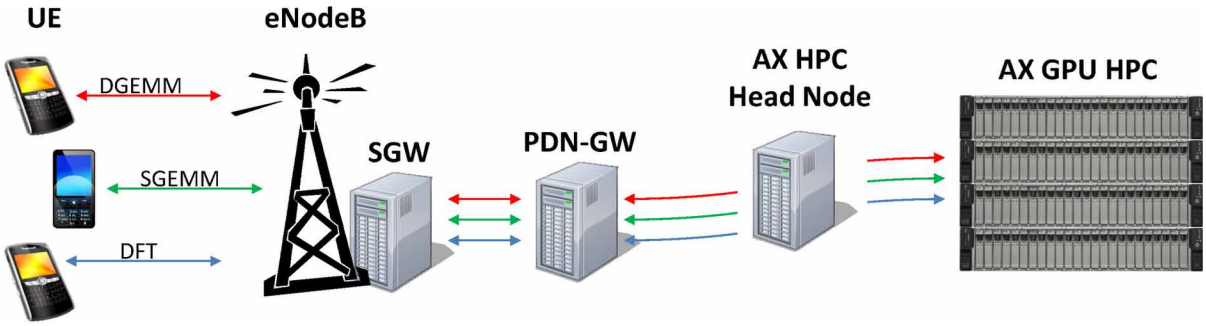
TSP VIEW OF AXaaS

The TSP could be considered as the arbiter of resource sharing and task allocation. By purchasing Computing platform instances from cloud operators (e.g., AWS (Amazon EC2, 2013)), or using their own resources (e.g., Terremark (Verizon-Terremark)), a meta-device can be established whose sole purpose is to execute generic Subroutines such as matrix multiplication in support of new compute-intensive applications. AX Requests called through an API layer need only be routed through the existing wireless broadband Network to an instance of AXaaS hardware. When speaking of the TSP, we assume a tier 1 network provider, due to their centrality to the internet backbone, along with their freedom from having to purchase transit services. This does not omit lower tier ISPs as potential AXaaS providers, though they may be faced with additional transit costs and larger communication overheads, thereby limiting their effectiveness in providing AX services.

Resource Acquisition

The Service Level Agreement (SLA) defines a commitment by the cloud providers to provide customers with consistent and reliable accessibility to purchased resources. This is centered on *availability* and *operational up-time*. AXaaS calls for an extension of the SLA to provide consistent and reliable *performance* in the acceleration of processes, which is at the heart of what AXaaS aims to provide. Quality of

Figure 6. The AXaaS model showing the basic AX transport scheme (AX Request and Replies). User Equipment (UE) communicates with LTE nodes (eNodeB) which tether to the IP backbone via Serving and Packet-Data Network Gateways (SGW/PDN-GW). IP traffic is then routed to the nearest AXaaS Head node which delegates resources in the GPU cluster to process AX requests. AX replies are routed via the same path.



Service (QoS) provisions seek to ensure the efficiency, speed and reliability of communication between entities over networks. Due to the nature of the proposed service model, QoS must also play a crucial role in ensuring maximum performance.

Since we envision the TSP as a broker, the AXaaS model contains two distinct layers: The first is the relationship between the TSP and cloud provider(s) in which resources are allocated, configured, monitored and maintained. Contractual agreements at this level (SLA and QoS) would have to be sufficient for the TSP to satisfy the aggregation of terms set forth on the next level, which is the relationship between the TSP and users. The users' expectations for accessibility and performance must be met in order for subscription and the use of accelerated applications to be advantageous. Separating the service model in this way allows each player to focus on their own area of expertise: cloud providers focus on maintaining their infrastructure on behalf of the TSP, and the TSP focuses on connecting subscribers to resources and handling the communication of data.

As an example formation of an AXaaS cloud, we consider Amazon Web Services who offers flexible, configurable instance types (Amazon-EC2-Pricing; EC2-Instance-Types). AXaaS applications would require high compute-capable instances with GPUs. We suggest a cluster of G2 (GPU compute) instances which can be launched into a common placement group with low latency networking. For our cost analysis we will use instances launched in the U.S. East Region, and specify the instance types as Heavy Utilization Reserved (EC2-Reserved-Instances). These instances have high availability and lower long term costs for the TSP, which can be calculated using the AWS Simple Monthly Calculator. Included are one-time Reserved Instance fees, monthly EC2 instance fees and estimated Data-Out transfer fees for 25 TB. Altogether, this cluster would theoretically allocate a peak 1.3 dEFLOPs (double-precision Exa FLOPs) or 30.1 sEFLOPs (single-precision Exa FLOPs) per month to a subscriber pool at a cost of $3,089.80 to the TSP.

Infrastructure and Personnel Costs

We presume that all cloud-side infrastructure maintenance is handled by the cloud provider in accordance with an SLA. The placement of the instances used for AXaaS must be considered with a perspective on application performance, not just capacity. It would be highly recommended to dedicate all instances within an AXaaS cluster to servicing only AXaaS requests. Programming the clusters for maximum performance requires the employment of various abstraction levels for API subroutines. We have shown previously that the use of these libraries (e.g., CUBLAS) leads to acceleration. Integrating applications with these libraries will be left to the AX developers. TSPs will need to invest in a software team to maintain these libraries and their interfaces for the developers.

Tiered Service Offerings

We conceptualize that the TSP offers AX in tiers using operation counts (FLOPs/mo), bundled with the existing data-throughput plans (GB/mo). Since the plan model graduates in user cost, we assert that it must also graduate in observable performance. We call upon the γ metric we introduced in Equation to assume this function, and frame our tier model around it. We suggest that each higher tier incorporates increasing QoS assurances for network performance (NQoS) and computation performance (CQoS), both of which limit the achievable values for γ. NQoS may be mitigated using various QCI Priorities for AX traffic (3GPP-TS23.401, 2008), via a subscription Tier identifier on the user's mobile device.

Table 2. Subscriber AX plan tiers showing subscriber cost, allocated operation count, Compute-Quality of Service guarantee (CQoS) and Network-Quality of Service guarantee (NQoS).

Tier	I	II	III	IV	V
Monthly Fee (F)	$10	$15	$30	$50	$100
TFLOPs Allocation (T)	10	20	40	80	160
GFLOPs/s (CQoS)	80	200	2700	14K	27K
Mbps (NQoS)	5	10	25	50	100

Its function would be to ensure a network transfer rate at least as high as that which is specified. CQoS may be controlled using the same Tier identifier, which would channel AX requests to appropriate GPU instances in the cloud. The quantification of subroutine performance would necessarily be a developmental issue, requiring some testing on the basis of application and platform. An example tiered plan is outlined in Table 2 with an outline of the evolution of advantages.

Tier I: Baseline tier for the casual user offering sufficient acceleration for AX Applications.
Tier II: Same as Tier I, except 2x volume and rate (high-end user).
Tier III: Much higher computational volume and assured network rate, permitting much higher γ values.
Tier IV: Computational volume, and assured network rate, along with γ increase substantially.
Tier V: Maximum computational volume and assured network rates, meant for high profile subscribers with exceptionally demanding applications.

ROI Analysis

To offset the cost of a cluster, the TSP must consider the sum of monthly fees and number of subscribers for each tier. If there are *N* tiers, and each tier *n* has *S* subscribers paying a monthly fee of *F* and the cluster presents a monthly cost of *C*, the gross monthly revenue R generated by the cluster can be calculated using Equation 3. For profitability, $R>0$ must be satisfied.

$$R = \sum_{n=1}^{N} \left(S_n F_n \right) - C \tag{3}$$

As a general principle, we can relate cluster computation allocation similarly. If *T* is the monthly TFLOPs allocation for each tier *n*, and *X* is the maximum theoretical TFLOPs that the cluster can compute monthly, we describe the percent allocation $Alloc_{\%}$ in Equation 4.

$$Alloc_{\%} = \frac{100}{X} \sum_{n=1}^{N} (S_n T_n) \tag{4}$$

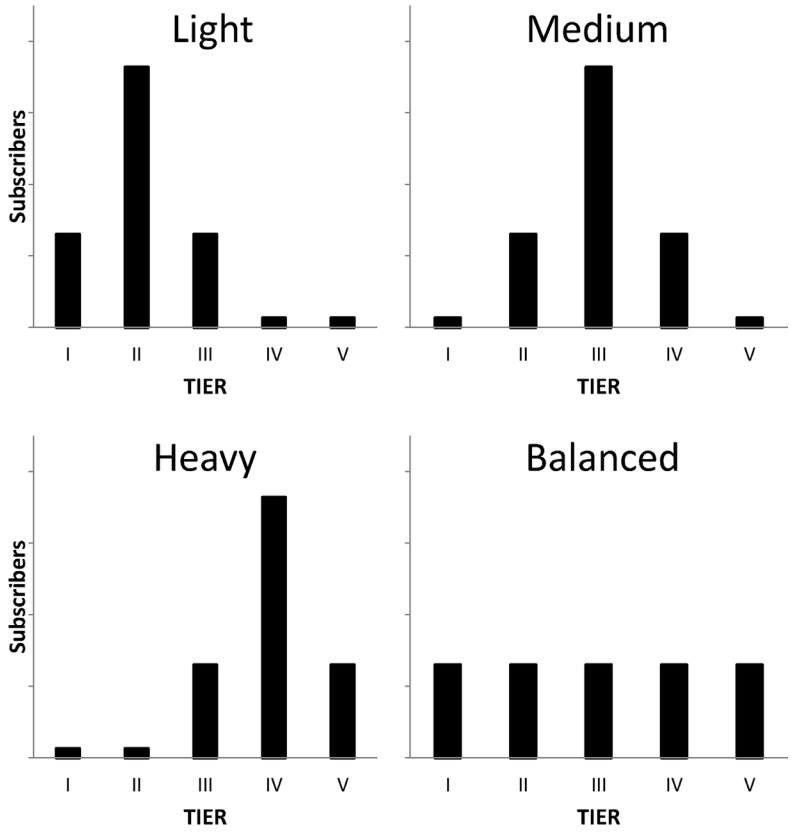
To provide an example of the effect of tier distribution we will take four distribution types: *Light*, *Medium*, *Heavy*, and *Balanced*. Refer to Table 3 and Figure 7 which suppose there are 27,000 subscribers to a cluster with a maximum of 1.3M available double-precision TFLOPs per month (i.e., 1.3 dEFLOPs).

Table 3. Allocation and Revenue effects of Tier Distributions.

Distribution	Light	Medium	Balanced	Heavy
Monthly Revenue	$510K	$850K	$996K	$1.4M
Comp. Allocation	56.0%	98.7%	134%	184%
Gross Profit @ 27K	99.4%	99.6%	99.7%	99.8%
Marginal Sub. #	164	99	84	62
Users at 100%	48K	27K	20K	14.6K
Revenue at 100%	$909K	$850K	$737K	$734K
Profit at 100%	99.7%	99.6%	99.6%	99.6%

Taking the *Medium* distribution as an example, we see that the TSP would need at least 99 subscribers to the cluster to break even (Marginal Sub. #). We also see that 27,000 subscribers in a Medium distribution across the tiers allocates 98.7% of the cluster compute capability in TFLOPs per month, and would generate approximately $850K in revenue at a gross margin of 99.64%. Focusing subscribers at the lower-usage level results in less impressive revenues for a given subscriber count, however leaves computational headroom for a greater number of additional subscribers. At 100% allocation, the *Light* distribution offers the best of revenues and profit margins.

Figure 7. Visualization of Tier Distributions in Table 3.



In all cases we are seeing impressive potential revenues and favorable gross margins. It is clear that launching a greater number of clusters to service even more subscribers in multiple regions will result in increased architectural complexity, but also much greater revenues. This section is not meant to be an exhaustive ROI analysis, but rather an initial demonstration of the potential of the AXaaS model. A complete ROI assessment will most definitely include many other considerations, including personnel expenses, which is beyond the scope of this chapter.

END-USER VIEW OF AXaaS

The impetus for a consumer to purchase a product involves its relative utility and cost. If there is an awareness of new significantly useful applications and services, the next question almost invariably is, "how much do they cost ?" The likelihood of a user to continue using a particular product relates to how well it performs and satisfies the user's needs. Useful applications that perform poorly may quickly be relegated to the dust bin. Subscription to AXaaS could be made quite easily by selecting it as an enhancement to an existing data plan. Since each device on the TSP network is identified by its mobile phone number, and is unique and wholly known by the TSP, device registration would be simple. A painless registration experience on the part of the user would enhance overall satisfaction with the service.

Energy Consumption Implications of AXaaS

Users prefer mobile applications with low energy consumption. Since computation requires energy, and our application requires large amounts of computation, we seek a quantifiable understanding of the impact of AX-enabled applications on the energy consumption of a mobile device. We will calculate the power consumption of AX enabled applications in two different scenarios.

- **Mobile-Device-Only Energy Consumption:** We will consider the iPhone 5S, whose battery capacity is approximately 6 Wh (1,570 mAh) (iPhone-5s), or 21,600 Joules. We estimated previously, the iPhone 5s processed a single FR routine in 4.6 seconds (i.e., γ=0.22). Reported by (Moto-X), the Apple A7 processor consumes 520 mA during floating-point operations (1.98 W @ 3.8 V battery voltage). This implies that the device would consume 9.09 Joules of energy in this 4.6 s face recognition interval. Therefore, the 21,600 Joule battery would be sufficient for 3 hours of continuous face recognition at the mere rate of γ=0.22, which will not generate any significant amount of interest for this application.
- **Mobile-AX Energy Consumption:** Let us calculate the energy consumption in the AX-enabled setting. In this scenario, the mobile device would still consume energy to capture an image from its camera, send it to the cloud via the telecom link, and receive the results. We will pick the 4G scenario in Figure 4 (10 Mbps and 100 ms). To process the image in Figure 1 (K=14), the phone would require 250 mJ of energy (116 mJ to send the 145KB image over the 4G link, during which the power consumption is 1W (Zhang, et al., 2010), and 50 mJ for image capture on highly-optimized hardware, and 84 mJ for idle time and to receive the results). This translates to γ=42 at a 0.75 W power consumption, and a battery life of 8 hours for continuous operation. These results show another favorable property of the AX-based acceleration: The *power efficiency* of the application changes from $9.09 W/\gamma$ (i.e., Watts per γ) to $18 mW/\gamma$, a 500x improvement (i.e., 1.98/0.22 vs. 0.75/42, respectively).

Application Access

Familiarity drives accessibility for a user. We suggest an access method that differs in no part from that which already exists. AX-enabled applications may be listed in their own category and be visible to all users regardless of their AXaaS subscription status. It is important to be able to investigate the range of possibilities before making a decision to subscribe. Non-subscribers who attempt to download an AX application could be redirected to an informational site on the TSP network which describes AXaaS and the ways subscription is necessary to power the application. After subscribing, the entire experience should be absolutely transparent. Besides having the ability to download and execute AXaaS applications, there should be no other indication that anything different has occurred. All application interactions should happen through the usual user interface, and the AXaaS performance guarantee would ensure a seamless experience.

Acceleration in Tiers

Since acceleration is tiered and priced by number of computations per month, the user must consider application use as an exhaustible resource. This way of thinking is quite familiar, as most data plans involve a finite capacity for data transfer. What the user would see in order to evaluate the economy of AXaaS is the number of AX requests they expect to be able to perform per month. This exact unit of measure could differ among TSPs. We suggest that this be a highly visible feature in each application (i.e., an AX-meter), integrated by developers, meant to enable a clear understanding of the application's consumption of billable computation. For users with multiple AX applications, which may not necessarily consume the same amount of computation per request, the client-side API layer must keep track of all usage and report that to the device so all applications may be updated accurately.

DEVELOPER VIEW OF AXaaS

Developers intent on providing state-of-the-art applications must immediately see the benefit of AXaaS. One simplified perspective is that applications reach a point of execution in which a function or group of functions are launched in a computational "black box" that resides in the cloud (AX servers). Since this "black box" consists of CUDA capable GPUs, there exists an array of device API libraries already authored and optimized. Client-side API libraries can be developed in order for mobile applications to remotely invoke function calls in the AX servers. A generalized model is desired for acceleration in which high abstraction subroutines such as CUBLAS (Nvidia CUBLAS) or CUFFT (Nvidia CUFFT) are readily available supporting a broad range of data types and sizes.

Acceleration API (AXAPI)

Considering the acceleration of FR, we begin when a mobile device has performed detection on a source frame from its camera and applied the requisite pre-processing. At this point projection must be offloaded to the cloud. In order for this to occur, the mobile device and the cloud must speak a similar language. In the context of FR, the mobile device must pass the test face data to the cloud, along with instructions specifying the type of process that should be executed on that data.

We suggest the employment of an AX-specific application programming interface (AXAPI) in order for seamless communication and interpretation of data and instructions between devices and the cloud. This would enable the mobile device code in our example to package and upload the test face data, along with an instruction for the cloud to perform GEMM in a single function call. It is obvious that the cloud must be configured and programmed to execute GEMM, and must also be preloaded with the database and Eigenvector. We suppose the Eigenvector is small enough to permanently reside in GPU Global or Constant memory, and is therefore immediately accessible to the GEMM kernel. Supposing the cloud software is ever aware of its resource utilization, it may select the most appropriate device and structure by which a kernel can be launched. The projection vector would then be calculated in a matter of milliseconds. Once a result is determined, it would be formatted and packaged for the return trip to the device which launched the request.

Ranging applications may involve the use of various data types such as single-precision float, complex or integer as well as varying provisional bit depths and decimal resolutions. The implication of the data type and bit depth is the time in which the computing device will take to execute the process. Reducing the bit depth of data by a factor might theoretically reduce the memory bandwidth usage and kernel execution time. Since Nvidia GPUs are inherently optimized for floating point calculations, AX calls with integer data types may in fact be better handled by CPUs, depending on the nature of the problem. General performance may be optimized by having AXAPI route calls to either GPU or CPU hardware such as the Xeon Phi coprocessor (INTEL-XeonPhi), depending on the characteristics of the data to be computed and the nature of the algorithm handling the computation. The use of the AXAPI to enable accelerated applications would simplify exploitation of cloud resources and allow the remaining application functions that do not require acceleration to be executed on mobile devices. We have outlined an example scenario in our background section, in which a mobile device uses acceleration to bypass its computational limitations in order to run a real-time FR application.

Application Framing

Designing applications to take advantage of AXaaS would involve integrating API layer function calls (i.e., AXAPI, as we described previously) at appropriate locations within application code. Having standardized client-side and host-side AX libraries is imperative for ensuring the ease of integration. Wrapping CUDA library functions into a higher abstraction layer would facilitate the entire process. Adapting code to the underlying architecture must be given special attention. Establishing a environment containing tier-controlled delegation of subroutines with varying performance expectations must also be considered in order to reinforce the efficacy of the performance-based tiered plans established by the TSP.

To preserve ease of integration, we imagine the developers to expect minimal modifications of code to enable process offloading. For example, the OpenCV object detection function *detectMultiScale* (OpenCV-FaceRecognizer) could be executed remotely by wrapping the function and its arguments within an AX context. Function *AXdetectMultiScale* in AXAPI might be sufficient by packaging the function arguments along with a function identifier for the AX cloud. The idea is to allow the feel of calling functions locally, with the resulting code executing remotely.

There are some potential third-party GPU cluster development tools readily available to provide a seamless integration of CUDA within a high performance cluster (HPC). While still under development, one possible solution is rCUDA (Duato, Pena, Silla, Mayo, & Quintana-Orti, 2011), which is a middleware aiming to virtualize available CUDA resources in a cluster and offers competitive performance and

efficient resource usage over low latency networks. rCUDA seeks to enable the remote exploitation of GPU hardware through the use of an API layer, which is the exact framework we seek for AXaaS. Recent work in the matter by (Sait & Vijayalakshmi, 2014) has shown excellent performance of the virtualized platform with respect to native performance.

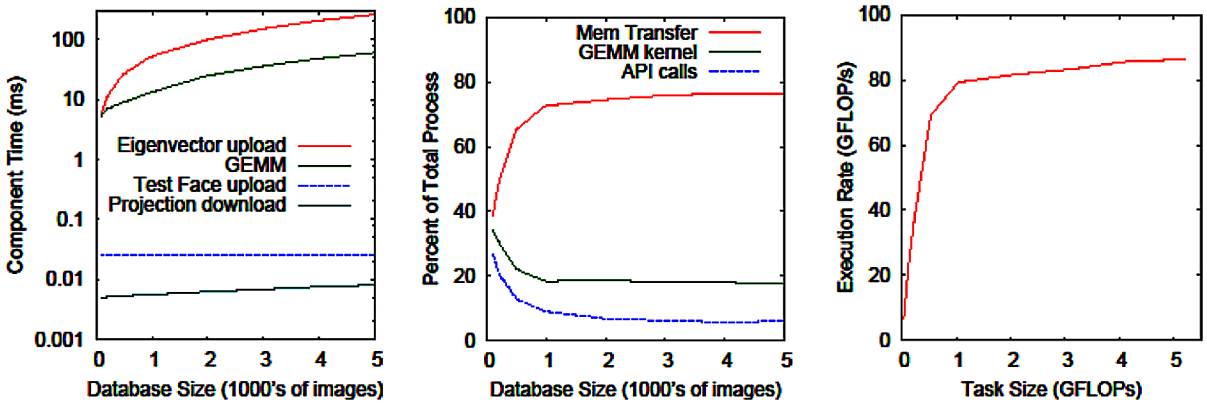
Transporting the Application Data

Figure 8 shows the nature of each component of the Projection function of the FR application on the GTX760 GPU. Our current FR projection method involves a large communication overhead across the PCI bus. Nearly 75% of the total projection time is attributed to memory transfer from host to device. In most cases, approximately 99% of all data transferred to device memory is the Eigenvector array. The projection vector generated by the multiplication is also relatively small and contributes minimally to the overall process time. Considerations such as this must be given during the design of applications framed by an acceleration methodology.

Even though the execution time of GEMM is proportional to the database size in our experiments, we do not see execution rate reflecting the intrinsic numerical power of the device until the data set grows large. The theoretical peak of GTX760 is 2258 single-precision GFLOP/s and 86.8 double-precision GFLOP/s. Figure 8 shows that the rate at which computations are performed by GEMM grows rapidly with the number of computations (database size) and appears to approach 86 GFLOP/s, which is 98% of the device's theoretical peak.

None of these results are particularly surprising. What they do provide is a reliable performance benchmark and insights into potential optimizations. Currently, GPU programmers are used to these concepts of 1) shuttling the application data back and forth between the host (CPU) and the device (GPU), 2) the consideration of the compute capability and double vs. single precision computation peaks, 3) GPU memory bandwidth. The AXaaS is a natural expansion of these concepts, where the GPUs reside in the cloud and should be fairly easy to get used to in terms of application development.

Figure 8. Duration and ratio of Projection call components (left), and execution rate of GEMM on the Nvidia GTX 760 (GK104).



Granularity of AX Requests

The smallest computational quantum in our FR application is the executing GPU kernel (Kirk & Hwu, 2010). Each kernel exists independently of all others and requires no cross-communication as a projection relies on unique data (the test face). We consider acceleration as a multi-user environment involving structured, asynchronous calls to specific acceleration functions. Maximizing performance for multiple users would involve spreading requests across streams and devices in such a way that will avoid queuing. One could consider larger, more complex AX request types that would benefit from finer grained parallelism, in which the process is spread across streams by batching kernels or even across multiple devices, depending on the breadth of the calculation. By applying a heuristic of task and data parallelism, the architecture of the GPU cluster can be efficiently profiled and utilized by the developer in order to provide minimal response times. Data parallelism will affect the occupancy of computing devices on a request basis. Task parallelism may be achieved with a sensitivity to the effects of occupancy and immediate resource availability.

Distribution

The marketplace for AXaaS applications should have visibility to TSP/AX subscribers alone. It would not make sense to advertise architecture-dependent applications to users without front-end access to that architecture. Usual distribution methods such as iTunes, Google Play or the Windows Phone App Store might be integrated with a AX-awareness that filters the list of displayable applications according to the AX subscription status of the browsing device in order to control for illegitimate downloads. Alternatively, applications that are downloaded to a non-subscriber's device could be directed by the API upon launching to the TSP AXaaS subscription domain. Another solution may be that the TSP hosts a proprietary AX application repository in which all TSP subscribers may browse and research applications, but only AX subscribers may purchase and download.

PERFORMANCE EVALUATION

In this section, we will evaluate the performance of AXaaS in a simulated mobile-cloud platform.

Experimental Setup

We identified the performance of individual FR subroutines by profiling our FR application using Nvidia Nsight Visual Studio Edition (Nvidia-VisualStudio). The platform used was a 64-bit Windows 7 Professional workstation with one Intel Core i7-4770K CPU @3.50GHz, 32GB RAM and an NVIDIA GTX 760 (GK104 architecture) GPU. The application was authored in C++ and compiled using Visual Studio 2010, OpenCV Library 2.4.8 and CUDA Toolkit v5.0. To supply the FR application with image data, we used an LG Nexus 4 running Android 4.2 Jelly Bean which can perform face detection using a built-in API, although it is only capable of detecting up to two faces per frame. Meant to support camera focusing, such a capability has uses for FR, but significantly limits the potential scope of the application (i.e., maximum achievable γ). Therefore, the mobile device was mainly used for sourcing frames in our experiments.

Evaluation Criteria

The central focus of our experiments with regard to acceleration was the matrix multiplication (GEMM) component of the Projection routine, since this phase of the FR application dominates the execution time. We had to consider memory bandwidth usage, task size in terms of number of computations and the kernel execution rate. The GTX760 GPU we used in our experiments have nearly identical single- and double-precision floating point performance to the Nvidia GRID K2 units used in the AWS G2 cluster, as shown in Table 4. Therefore, we used the GTX760-based results to estimate the AWS G2 instance performance. We extrapolated our results to the GTX Titan Z GPU in the third column of Table 4 when estimating the performance for tier III, IV, and V operation. We expect the TSP to utilize the lower end GPUs only for acceleration types requiring heavy single-precision due to its lower cost, while GPUs like GTX Titan Z will be necessary for parts of the code that operate on heavy double-precision floating point data, such as the Projection phase of the FR application.

Experimental Results

Table 5 shows measured Projection performance for the GTX760 along with interpolations for the GRID K2, hence leading to our estimations for the AWS HPC (HPC-Servers). Our estimations incorporate optimizations such as: 1) an omission of the eigenvector upload on each kernel launch, since a single upload can be amortized over a vast number of FR requests and 2) a full concurrency, where up to 16 independent streams may be executing simultaneously, which is a realistic assumption for a cloud environment utilizing heavy parallelism. Observe that a fully saturated GRID K2 can theoretically support a computational throughput of 200 FR Projections per second on a 5000-image database. A cluster of five K2 devices augments this to roughly 1000 FR projections per second (last row) for the same database.

Table 4. Comparison of GPU performance and architectures.

GPU	GTX 760	GRID K2	GTX Titan Z
Architecture	GK104	GK104	2 x GK110
CUDA Cores	1,152	1,536	2 x 2,880
GPU Clock	980 MHz	745 MHz	705 MHz
Memory Size	2 GB	4 GB	6+6 GB
GFLOPS (single)	2,258	2,288	8,122
GFLOPS (double)	94	95	2,707

Table 5. GK104 double-precision GEMM performance; Projection times and estimated executable projections per second.

Database Size	1000	2000	3000	4000	5000
760 PJ Time (ms)	71.9	132.3	199.1	269.9	339.9
760 Stream #PJ/s	13.9	7.6	5.0	3.7	2.9
K2 PJ Time (ms)	19.6	33.7	48.1	63.7	80.1
K2 Stream #PJ/s	51.0	29.7	20.8	15.7	12.5
K2 Device #PJ/s	815.8	475.1	332.6	251.4	199.6
(5) K2 Cluster #PJ/s	4079	2376	1663	1257	998.1

Table 6. FR performance for a 5000-image database, for Tiers I, II (K=14), and Tiers III–V (K=80), and predictions for a decade later.

Tier	I	II	III	IV	V	Future
TFLOPs / mo	10	20	40	80	160	400
dGFLOP/s CQoS	80	200	2700	14K	27K	100K
Mbps NQoS	5	10	25	50	100	1000
Nwk Time (ms)	398	199	618	309	154	18.6
Exec Time (ms)	964	386	162	33	16	4.3
Req Time (ms)	1362	585	780	342	170	22.9
K (faces/frame)	14	14	80	80	80	80
FR γ	**10.9**	**25.1**	**103**	**234**	**469**	**3493**

Table 6 provides a list of AXaaS tiers with their network bandwidth and computational limits in the way previously described. This table is an extrapolation of the results provided in Table 5. Tiers I and II are assumed to utilize the Grid K2 GPU devices, and provide a network bandwidth guarantee (NQoS) of 5–10 Mbps, achieving γ values in the 10–20 range, which is perfectly sufficient for the casual user. Tier II allows 2x the allocate-able FLOPs, thereby allowing the user to enjoy 2x more runtime for the month of subscription. The "power tiers" (III–V) achieve much more impressive γ values in the 100–500 range and allow the user 16x more runtime for the month of subscription (i.e., 160 TFLOPs for tier V vs. 10 TFLOPs for tier I).

Our final column looks forward, perhaps a decade into the future. Assuming that LTE-Advanced takes flight, offering full 1 Gbps transfer rates, and considering a cluster of 25 GPU servers, we estimate a potential $\gamma > 3000$. Within this time frame, an advanced mobile device such as iPhone is only expected to reach γ values of 1–2 from its current 0.22 based on its past decade's trend. Note that these results are meant to be no more than a back-of-the-envelope analysis to show the long term business viability of AXaaS, rather than a complete business study. Final tier layouts of the product will entirely depend on the resources and business model of the TSP.

CONCLUSION

We proposed a new service architecture called Acceleration as a Service (AXaaS) whose central focus is on accelerating specific portions of advanced mobile application code through cloud-based augmentation. AXaaS is formulated to enable a new generation of resource-intensive mobile applications such as real-time face recognition (FR), real-time language translation and augmented reality. When executed on mobile devices alone, hardware limitations of mobile devices prevent these applications from achieving satisfactory performance levels today. This fact is not expected to change in the foreseeable future considering the mobile architectural improvement trends. Our conceptualization of the AXaaS model is based on our observation that, these resource-intensive mobile applications spend a vast majority of their execution time on a very small set of API functions such as generalized matrix-matrix operations (GEMM) or Fast Fourier Transform (FFT) operations. Therefore, to significantly improve application performance, only these API functions need to be executed much faster (i.e., *accelerated*) instead of the entire application.

We identified Telecom Service Providers (TSPs) as the most logical candidate for providing *acceleration* (abbreviated AX) as a Service (aaS), hence our naming AXaaS. We showed that AXaaS could provide a new lucrative revenue stream for TSPs when offered in different tiers, similar to today's tiered data services. By renting resources from cloud providers, we showed that TSPs could generate favorable gross profit margins from AXaaS subscriptions. We demonstrated that the AXaaS model is scalable: When a large customer base is pooling AX computational resources, they do not experience performance degradations as long as TSPs provision their hardware resources accordingly.

We also provided a study of how AXaaS affects a developers' application programming environment. We showed that the AXaaS programming model (through an interface we termed AXAPI) is very similar to today's GPU programming models with minor modifications, thereby presenting no major barrier for entry to developers. Furthermore, we estimated that, AXaaS-based computation offloading dramatically improves the energy consumption of mobile applications by up to two or three orders-of-magnitude, thereby providing a great avenue for developers to create energy-efficient applications.

Our major focus was the impact of AXaaS on user experience. We defined a metric γ that quantifies user experience, which is the *total number of faces recognized in a second*. While this metric is specific to the real-time FR application, we observed that, such a metric is possible to define for every application that can be augmented by the AXaaS model. For example, real-time language translation can use a similar metric called *sentences translated per second*. It is also clear that the γ metric is directly related to performance, or specifically, the degree of acceleration. User experience will be completely different for different γ values, which we captured through our formulation of "acceleration tiers." In the case of real-time FR, a standalone iPhone 5s can reach γ=0.22, which is barely useful, if not flat out boring. When acceleration is applied, γ=10-20 can be achieved in tiers I and II, which will turn this application into a useful one, while γ=100 in tier III will make the application wholly exciting. On the other hand, γ=200-500 in tiers IV and V will allow the application to be used in demanding organizational scenarios such as real-time surveillance.

While application coding will face minimal changes across different tiers, varying degrees of acceleration in different tiers will completely change the application's relative value and utility. So, from the standpoint of the TSP, different values of γ present different tiers of marketable products which different user groups will be willing to pay for. We showed in our experimental results that, γ=469 is feasible on today's 4G networks, and γ>3000 can be achieved when LTE Advanced becomes commonplace within the next decade. Therefore, we conclude that AXaaS is a novel and viable business model today and will continue to be in the foreseeable future.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation grant CNS-1239423 and by Nvidia Corp.

REFERENCES

Alling, A., Powers, N., & Soyata, T. (2015). Face Recognition: A Tutorial on Computational Aspects. In *Emerging Research Surrounding Power Consumption and Performance Issues in Utility Computing*. IGI Global.

Amazon EC2. (2013). Retrieved from Retrieved from: http://aws.amazon.com/ec2/sla/

Amazon-EC2-Pricing. (n.d.). Retrieved from http://aws.amazon.com/ec2/pricing/

Andoid-API. (n.d.). *Android API Reference*. Retrieved from http://developer.android.com/reference/

Microsoft Azure. (2014). Retrieved from Microsoft Azure: http://azure.microsoft.com/en-us/support/legal/sla/

Bradski, G., & Kaehler, A. (2008). *OpenCV Computer Vision with OpenCV Library*. O'Reilly.

Chen, E., Ogata, S., & Horikawa, K. (2012). Offloading android applications to the cloud without customizing android. *Pervasive Computing and Communicatiosn Workshop (PERCOM Workshops)* (pp. 788-793). IEEE. doi:10.1109/PerComW.2012.6197619

Cho, J., Mirzae, S., Oberg, J., & Kastner, R. (2009). FPGA-based face detection system using Haar Classifiers. *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, (pp. 103-112). doi:10.1145/1508128.1508144

Chun, B. G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud. *Proceedings of the Sixth Conference on Computer Systems*, (pp. 301-314). doi:10.1145/1966445.1966473

Costa, P., Migliavacca, M., Pietzuch, P., & Wolf, A. (2012). NaaS: Network as a Service in the cloud. *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE '12).*

Cuervo, E., Balasubramaniam, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). MAUI: Making smartphones last longer with code offload. *Proceedings of the 8th international Conference on Mobile Systems, Applications, and Services* (pp. 49-62). ACM. doi:10.1145/1814433.1814441

Dsouza, A., Kabbedjik, J., Seo, D., Jansen, S., & Brinkkemper, S. (2012). Software-as-a-service: Implications for business and technology in product software companies. *PACIS 2012 Proceedings Pacific Asia Conference on Information.*

Duato, J., Pena, A. J., Silla, F., Mayo, R., & Quintana-Orti, E. S. (2011). Performance of cuda virtualized remote gpus in high performance Clusters. *International Conference on Parallel Processing (ICPP)* (pp. 365-374). IEEE. doi:10.1109/ICPP.2011.58

EC2-Instance-Types. (n.d.). Retrieved from http://aws.amazon.com/ec2/instance-types/

EC2-Reserved-Instances. (n.d.). Retrieved from http://aws.amazon.com/ec2/purchasing-options/reserved-instances/

Emarketer. (2014). *Smartphone Users Worldwide Will Total 175 Billion 2014.* Retrieved from http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536/

Gentry, C. (2009). *A Fully Homomorphic Encryption Scheme*. Stanford University.

Gentry, C., Halevi, S., & Smart, N. (2012). *Homomorphic evaluation of the AES circuit* (pp. 850–867). CRYPTO.

Google-Translate. (n.d.). Retrieved from https://translate.google.com/

3GPP-TS23.228. (n.d.). *IP multimedia subsystem (ims); stage 2 (release 9).* Retrieved from 3rd Generation Partnership Project, Tech. Rep., 2010, 3GPP TS 23.228 V9.3.0 (2010-03).

3GPP-TS23.401. (2008). General packet radio service (gprs) enhancements for evolved universal terrestrial radio access network (e-utran) access. Release 8. Retrieved from 3rd Generation Partnership Project, Tech Report 3GPP TS 23.401 V8.1.0.

Halevi, S., & Shoup, V. (2014). Algorithms in HElib. *Proceedings, Part I, Advanced in Cryptology - {CRYPTO} 2014 - 34th Annual Cryptology Conference*, (pp. 554-571). Santa Barbara, CA. doi:10.1007/978-3-662-44371-2_31

HPC-Servers. (n.d.). *High Performance Computing for Servers — Tesla GPUs*. Retrieved from http://www.nvidia.com/object/tesla-servers.html

Huang, J., Qian, F., Gerber, A., Mao, M., Sen, S., & Spatscheck, O. (2012). A close examination of performance and power characteristics of 4G LTE Networks. *Proceedings of the 10th international conference on Mobile systems, applications, and services* (pp. 225-238). ACM. doi:10.1145/2307636.2307658

INTEL-XeonPhi. (n.d.). *INTEL Xeon PHI Family*. Retrieved from http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html

iPhone-5s. (n.d.). *The iPhone 5s Specifications*. Retrieved from AnandTech: http://www.anandtech.com/print/7335/the-iphone-5s-review

Keller, J. (2011). *Cloud-Powered Facial Recognition is Terrifying*. Retrieved from http://www.theatlantic.com/technology/archive/2011/09/cloud-powered-facial-recognition-is-terrifying/245867/

Kim, K., Jung, K., & Kim, H. (2002). Face recognition using kernel principal component analysis. *IEEE Signal Processing Letters*, 40–42.

Kirk, D. B., & Hwu, W. M. (2010). *Programming Massively Parallel Processors*. Morgan-Kaufmann.

Kocabas, O., & Soyata, T. (2014). Medical Data Analytics in the cloud using Homomorphic Encryption. In *Handbook of Research on Cloud Infrastructures for Big Data Analytics* (pp. 471–488). US: IGI Global.

Kocabas, O., Soyata, T., Couderc, J.-P., Aktas, M., Xia, J., & Huang, M. (2013). Assessment of Cloud-based Health Monitoring using Homomorphic Encryption. *Proceedings of the 31th IEEE Conference on Computer Design*, (pp. 443-446). Asheville, NC, USA. doi:10.1109/ICCD.2013.6657078

Kovachev, D., Cao, Y., & Klamma, R. (2013). *Mobile Cloud Computing: A Comparison of Application Models*. Retrieved from http://arxiv.org/abs/1107.4940v1

Kwon, M., Dou, Z., Heinzelman, W., Soyata, T., Ba, H., & Shi, J. (2014). Use of Network Latency Profiling and Redundancy for Cloud Server Selection. *Proceedings of the 7th IEEE International Conference on Cloud Computing*, (pp. 826-832). IEEE. doi:10.1109/CLOUD.2014.114

Mei, C., Shimek, J., Wang, C., Chandra, A., & Weissman, J. (2011). *Dynamic outsourcing mobile computation to the cloud*. University of Minnesota.

Moto-X. (n.d.). *Google/Motorola Mobilitys Moto X Outpaces Competition*. Retrieved from https://www.abiresearch.com/press/googlemotorola-mobilitys-moto-x-outpaces-competition

MS-Translator. (n.d.). Retrieved from http://www.microsoft.com/en-us/translator/

NIST. FIPS-197. (2001). Advanced encryption standard (AES). National Institute of Standards and Technology.

Nvidia, *C. U. B. L. A. S.* (n.d.). Retrieved from https://developer.nvidia.com/cublas

Nvidia, *C. U. D. A*. (n.d.). *NVIDIA CUDA*. Retrieved from nvidia.com: http://www.nvidia.com/object/cuda_home_new.html

Nvidia, *C. U. F. F. T.* (n.d.). Retrieved from https://developer.nvidia.com/cufft

Nvidia, *C. U. S. P. A. R. S. E.* (n.d.). Retrieved from https://developer.nvidia.com/cusparse

Nvidia-Shield. (2014). *The Nvidia Shield Gaming Tablet*. Retrieved from http://shield.nvidia.com/gaming-tablet/

Nvidia-VisualStudio. (n.d.). *NVIDIA Nsight Visual Studio Edition*. Retrieved from https://developer.nvidia.com/nvidia-nsight-visual-studio-edition

OpenCV-FaceRecognizer. (n.d.). *Open CV 3.0.0-dev Documentation*. Retrieved from http://docs.opencv.org/trunk/modules/contrib/doc/facerec/facerec_api.html

Page, A., Kocabas, O., Ames, S., Venkitasubramaniam, M., & Soyata, T. (2014). Cloud-based Secure Health Monitoring: Optimizing Fully-Homomorphic Encryption for Streaming Algorithms. *IEEE Globecom 2014 Workshop on Cloud Computing Systems, Networks, and Applications.*

Page, A., Kocabas, O., Soyata, T., Aktas, M., & Couderc, J.-P. (2014). Cloud-Based Privacy-Preserving Remote ECG Monitoring and Surveillance. *Annals of Noninvasive Electrocardiology*, n/a. doi:10.1111/anec.12204 PMID:25510621

Powers, N., Alling, A., Gyampoh-Vidogah, R., & Soyata, T. (2014). AXaaS: Case for Acceleration as a Service. *IEEE Globecom 2014 Workshop on Cloud Computing Systems, Networks, and Applications.*

Sait, Y., & Vijayalakshmi, R. (2014). Enabling high performance computing in cloud infrastructure using rCUDA.

Sandikkaya, M., & Harmanci, A. (2012). Security problems of Platform as a Service (PaaS) clouds and practical solutions to the problems. *31st International Symposium on Reliable Distributed Systems*. doi:10.1109/SRDS.2012.84

Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *Pervasive Computing*, 14-23.

Soyata, T., Ba, H., Heinzelman, W., Kwon, M., & Shi, J. (2013). Accelerating mobile cloud computing: A survey. In *Communication Infrastructures for Cloud Computing* (pp. 175–197). IGI Global.

Soyata, T., Muraleedharan, R., Funai, C., Kwon, M., & Heinzelman, W. (2012). Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. *Computers and Communications (ISCC), 2012 IEEE Symposium on*, 59-66.

Soyata, T., Muraleedharan, R., Langdon, J., Funai, C., Kwon, M., & Heinzelman, W. (2012). COMBAT: mobile-Cloud-based cOmpute/coMmunications infrastructure for BATtlefield applications. *Proceedings of the Society for Photo-Instrumentation Engineers*, 8403.

Turk, M., & Pentland, A. (1991). Face recognition using eigenfaces. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1991. Proceedings CVPR*, *91*, 568–591.

Verbelen, T., Simoens, P., DeTurck, F., & Dhoedt, B. (2012). Cloudlets: Bringing the cloud to the mobile user. *Proceedings of the third ACM workshop on Mobile cloud computing and services* (pp. 29-36). ACM. doi:10.1145/2307849.2307858

Verizon-Terremark. (n.d.). *Verizon Terremark*. Retrieved from http://www.terremark.com/

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (pp. 511-518). doi:10.1109/CVPR.2001.990517

Viola, P., & Jones, M. J. (2001). Robust real time face detection. *Second International Workshop on Statistical and Computational Theories of Vision - Modeling, Learning, Computing, and Sampling*, (pp. 1-25).

Wang, H., Liu, W., & Soyata, T. (2014). Accessing Big Data in the Cloud Using Mobile Devices. In *Handbook of Research on Cloud Infrastructures for Big Data Anal* (pp. 444–470). IGI Global. doi:10.4018/978-1-4666-5864-6.ch018

Yang, M., Kriegman, D., & Ahuja, N. (2002). Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(1), 34–58. doi:10.1109/34.982883

Zhang, L., Tiwana, B., Qian, Z., Qang, Z., Dick, R. P., Mao, Z. M., & Yang, L. (2010). Accurate online power estimation and automatic battery behavior based power model generation for smartphones. Intl. Conf. on Hardware-Software Codesign and System Synthesis (CODES+ISSS). Scottsdale, AZ.

KEY TERMS AND DEFINITIONS

3G Network: A set of standards to define the third generation of mobile telecommunications technology. This standard is somehow outdated as of the publication of this book chapter.

4G Network: Fourth generation mobile telecommunications technology offering substantial improvements over the 3G.

5G Network: A telecommunications standard (fifth generation wireless systems) that is not yet available. This technology will offer data rates up to 1 Gbps.

Acceleration (AX): Speeding up an application by intelligently identifying the parts of the code that benefit from faster execution. In the applications introduced in this chapter, these parts of the code that can benefit from AX include FFT, and GEMM routines.

Augmented Reality: A generalized family of applications in which a computer-generated image is super-imposed on an acquired image from a camera. This super-imposed image is intended to provide more information to the user, such as, how a house would look when a refrigerator is placed in it.

AXaaS (Acceleration as a Service): A service offered by a TSP that is based on a monthly service charge. The product to rent is the acceleration of portions of the mobile code that a mobile user is running. In this chapter, it is determined that, the acceleration of a small subset of functions such as GEMM and FFT will be sufficient to drastically improve the performance of certain mobile applications, such as real-time Face Recognition.

Basic Linear Algebra Subroutines (BLAS): A set of API functions that are introduced in three levels. Level 1 BLAS API functions include vector-vector operations. Level 2 BLAS functions include vector-matrix operations, while Level 3 includes matrix-matrix operations.

Cloud Computing: A paradigm where computational and storage services can be purchased on demand from cloud service providers much like gas and electric can be purchased based on usage from utility providers.

Cloud Instance: A server of a storage unit that a cloud user rents from a cloud operator. Each cloud instance comes with a different pricing depending on the time commitment to use this resource, as well as the capability of this resource.

Cloud Operator: The business entity that invests in an infrastructure including multiple datacenters in a broad geographic area. This entity has multiple customers that rents cloud computing services to and has the advantage of pooling resources, thereby taking advantage of economies of scale.

Cloudlet: An intermediate device between a mobile device and the cloud with the purpose of accelerating a mobile application.

Computational Quality of Service (CQoS): Due to the emphasis on the computational intensity, a separate metric, CQoS is defined to quantify a guaranteed level of computational throughput.

Compute-Unified Device Architecture (CUDA): The programming language and the GPU architecture introduced by Nvidia Corp. to provide a complete GPU programming platform for general purpose scientific programmers.

CUFFT, CUSPARSE, CUBLAS: Three Nvidia CUDA based libraries with API functions designed to accelerate FFT, BLAS, and Sparse Matrix operations.

Face Recognition: A computationally-intensive application in which a face that is detected in a picture frame is compared against a set of database faces and a decision is made as to which face it matches with a high probability. This application is a perfect candidate for AXaaS and has been extensively studied in this chapter.

Fast Fourier Transform (FFT): A transformation that turns time-domain data into frequency-domain data, since performing the operations in the frequency domain result in a much lower overall processing time for certain algorithms such as Image Processing.

Generalized Matrix-Matrix Multiplication (GEMM): A family of matrix operations in which each matrix could actually be a vector. A vector could be thought of a matrix with a dimension of 1 in one of the axes.

Graphics Processing Unit (GPU): A device that is capable of processing information in a massively-parallel (MP) fashion. MP processing differs substantially from traditional parallel processing in that, GPUs are actually not good at processing a few threads, but, rather, tens or hundreds of thousands of threads. This makes GPUs suitable computational device for applications that are inherently massively-parallel, such as Image Processing, or general Digital Signal Processing.

IaaS (Infrastructure as a Service): A cloud computing paradigm, where the cloud service provider rents an entire infrastructure which may include servers, storage, and operating software.

Long Term Evolution (LTE): A high speed wireless communication standard that is reasonably widespread. It offers data rates up to 100 Mbps, whereas LTE Advanced promises rates up to 1 Gbps.

Mobile-Cloud Computing: A computing paradigm where the computational and/or storage power of a mobile device is substantially augmented by using cloud resources via a communications link.

Mobile-Cloud Offloading: A process where a part of the mobile application code is executed in the cloud.

Natural Language Processing (NLP): Another computationally-intensive candidate application for AXaaS in which the words that are spoken by a human is being somehow interpreted by a computer.

Network Quality of Service (NQoS): This metric is very close to the traditional QoS in that, a guarantee is defined on the network throughput between the mobile device(s) and the cloud.

PaaS (Platform as a Service): A cloud computing paradigm where the cloud service providers rents a platform, including programming languages, libraries, tools and services. Clients control deploying their applications and cloud providers manage the underlying cloud infrastructure based on demand.

Packet Data Network Gateways (PDN-GW): A Packet Data Network (PDN) Gateway provides connectivity from User Equipment (UE) to external PDNs.

Quality of Service (QoS): A traditional metric defined to quantify a level of *guaranteed* performance, such as bandwidth, computation, and more. In this chapter, this metric is broken down into two inter-related metrics.

Return on Investment (ROI): A metric that is introduced to quantify the amount of time that it would take to receive sufficient profits from an investment to reach overall profitability. After this point, the investment provides profitability going forward.

SaaS (Software as a Service): A cloud computing paradigm where the users run an application directly from the provider's web interface, thereby completely avoiding any investment on the infrastructure to run such software. An example is Microsoft Office 365.

Service Level Agreement (SLA): An agreement signed by the user and the cloud provider with the intent to agree on a set of performance metrics, such as bandwidth and uptime. AXaaS model adds a performance metric to traditional SLAs.

Tegra: A family of processors introduced by Nvidia that contain a CPU and GPU, with multiple cores within each sub-structure.

Telecom Service Provider (TSP): The provider of communication to users, which can offer voice as well as data services through the same exact network by adhering to well-defined standards, such as 3G, 4G, and 5G.

TFLOPS (Tera Floating Point Operations Per Second): A metric that defines how many trillion (Tera) Floating Point operations a computational device is capable of performing per second. Since floating point operations can be single, double, or even quadruple, there is a slight vagueness in this metric. To eliminate this ambiguity, in this chapter two separate metrics have been used. sTFLOPS is the single precision and dTFLOPS is the double-precision floating point operation quantity.

Tiered Pricing: A pricing model where different levels of the same product are offered with varying pricing. In the AXaaS model, this tier structure is based solely on a performance metric γ that quantifies the user experience.

User Experience (γ): A metric that is defined in this chapter to quantify the utility that a consumer is receiving from an AXaaS subscription. A TSP can formulate a pricing structure for the AXaaS based on such a metric. Higher values of γ imply being able to use the same application for much more sophisticated purposes, even if the application itself does not change.