Workload Characterization for OpenCL-based Mobile Offloading Framework

Heungsik Eom, Pierre St Juste, Renato Figueiredo
Advanced Computing and Information Systems Laboratory
Electrical and Computer Engineering
University of Florida
Gainesville, Florida, USA
Email: {hseom, pstjust, renato}@acis.ufl.edu

Omesh Tickoo, Ramesh Illikkal, Ravishankar Iyer
Intel Corporation
2111 N.E. 25th Avenue
Hillsboro, Oregon, USA
Email: {omesh.tickoo, ramesh.g.illikkal, ravishankar.iyer}@intel.com

***Abstract*—Despite the enhancements in mobile hardware platforms to meet the increasing computing demands of mobile applications, the inescapable fact is that the small battery capacity will always serve as a bottleneck hindering their processing capabilities. To address this challenge, workload offloading has been proposed in the research community as a viable alternative which enables mobile platforms to offload computational workloads to more powerful computing elements such as workstations and the cloud. However, in order to make these offloading techniques performance-improved and energy-efficient, it is important to have a proper framework that allows the dynamical choice between local execution and remote offloading based on certain conditions. In this paper, we characterize mobile workloads for the suitability of offloading from the perspective of computation to communication ratio measured by the time for workloads to be executed locally divided by the amount of data to be transferred for remote offloading. As part of the workload characterization effort, we developed a remote offloading framework by extending the well-defined hardware-level offloading standard, OpenCL framework, over the network using the TCP/IP stack and we strategically selected four representative OpenCL workloads, two computation-intensive and two communication-intensive workloads. Furthermore, by deploying the offloading framework into the various types of network such as local area networks, campus networks, and Amazon EC2 instance which have different network limitations, we analyze the behavior of the remote offloading framework in accordance with environmental factors such as network conditions or the computing capabilities of offloadable resources. Our experimental results indicate that the effectiveness of offloading mobile workloads to remote resources depends tightly on the workload characteristics and environmental conditions. We believe that the numerical results presented in this paper will guide a basis of designing the intelligent run-time offloading scheduler.**

I. Introduction

In recent years, mobile computing has become the preferred method of personal conputing for millions of users. In fact, the mobile device market has skyrocketed with over 400 million smartphones shipped in 2011 and an estimated prediction of 1 billion shipments for 2015 [1]. This growth has been spurred by availability of hundreds of thousands of mobile applications such as social networking, location-based services, image processing, augmented reality, face and speech recognition. In order to meet the increasing computing demands of these applications, mobile platforms have been augmented with multi-core CPUs, more powerful GPU, and other specialized hardware accelerators. Although the enhancements in mobile hardware platforms have significantly improved their processing capabilities, they have also exacerbated their energy limitations. As more complex hardware is added to mobile platforms, delivering optimal energy performance becomes an increasingly challenging problem. Furthermore, the small battery capacity will always serve as a bottleneck hindering the processing capabilities of these enhanced platforms. To address these issues, there are two main ongoing efforts being explored by the research community: heterogeneous computing and cloud offloading. Firstly, heterogeneous computing is a mechanism to provide various range of executions by dynamically combining processing elements having different performance and energy implications and opportunistically scheduling the workloads on the optimal processing element based on the workload requirements and energy availability [2]. Heterogeneous architectures with big/small cores, CPU/GPU and CPU/hardware accelerators are now being offered by leading processor/SoC vendors [3], [4]. Secondly, computation offloading has been proposed as an alternative for extending the capabilities of mobile devices by seeking intelligent ways to enable mobile devices to leverage the computing capabilities of more powerful resources such as privately owned workstations [5] or public cloud (i.e. Amazon EC2) [6] over the network. However, these studies lack an in-depth, collaborative investigation of the effect of workload characteristics and environmental factors such as network conditions and the computing capabilites of the remote resources into offloading performance and energy implication. In fact, in order to make these computation offloading techniques performance-improved and energy-efficient, it it important to have a framework that allows the choice between local execution and remote offloading so that those techniques are able to seamlessly and transparently offload workloads to networked

devices based on certain conditions.

In this paper, we characterize mobile workloads for the suitability of offloading from the perspective of *Computation to Communication ratio* which is calculated by the time for workloads to be executed locally divided by the data transfer time for the remote offloading. Thus in our work, computation to communication ratio for each workload is a comprehensive measurement which mirrors three parameters such as the volume of computation of workloads, the amount of data to be transferred, and the network conditions. As part of the workload characterization effort through computation to communication ratio, we developed a remote offloading framework by extending the well-defined hardware-level offloading standard, OpenCL framework, to leverage the various range of computing elements from general purpose CPUs to special hardware accelerators over the network using the TCP/IP stack. OpenCL is a framework specifically designed to offload workloads in heterogeneous platforms, however, it currently does not support the access to accelerators across the network. Although the proposed framework is not the first to propose the concept of offloading heterogeneous workloads over the network (i.e. Remote CUDA [7], Virtual OpenCL [8]), it is the first to consider this approach where offloading heterogeneous workloads happens between a mobile device and a virtual machine instance running in the cloud or a workstation across the Internet. After developing the OpenCL-based offloading framework, we strategically selected four quantitatively and qualitatively different OpcnCL workloads for the analysis: Sobelfilter, matrix multiplication, *N*-body physics and hidden Markov model, each being classified into two categories: the computation-intensive workload (matrix multiplication and *N*-body physics) and the communication-intensive workload (Sobelfilter and hidden Markov model) in terms of computation to communication ratio. Note that though computation- or communication-intensity of four OpenCL workloads is not absolute, it is worth to categorize the relative computation- or communication-intensity of the workloads so it is possible to explore the workload conditions where offloading is more beneficial than local processing.

Furthermore, we deployed the offloading framework into the various types of network such as local area networks, campus networks and Amazon EC2 instance which have different network restrictions, and set various levels of the computing capability of the remote resource from general purpose CPUs to Amazon EC2 GPUs cluster instance to analyze the behavior of the remote offloading framework in terms of the performance and energy implication of mobile devices in accordance with environmental factors such as network conditions and the computing capabilities of offloadable resources. For example, we configure local area networks by directly connecting a mobile device and a remote resource with Nvidia graphics processing units via a wireless router which represents the best resource of our experimental setups. The experimental results show that although not all types of workloads benefit from mobile workload offloading, there clearly exist a class of workloads and environmental conditions that can leverage the remote offloading.

The rest of the paper is outlined as follows. In Section II, we present the necessary background of heterogeneous computing and OpenCL. Section III describes basic components and their functions of mobile workload offloading system we implemented. We present our methodology for analyzing the behavior of mobile workload offloading and experimental results in Section IV and V, respectively. In Section VI, we overview previous related works on computation offloading framework. Lastly, Section VII concludes the paper.

II. Background

A. Heterogeneous Computing

An increasing number of hardware platforms are incorporating heterogeneity for both power and performance improvements. Therefore, we have seen the primary host CPU cores being assisted by various compute units for specialized function offload. These specialized accelerators are aimed at speeding up specific portions of the application code and differ from the general purpose processor architectures. In this section, we explain some of the different types of heterogeneous platforms available nowadays.

Heterogeneous Multicore CPU Combinations: In this design, high performing CPU cores are integrated with low performing power efficient cores. Currently, some mobile and server platforms employ this architecture [9]. The goal is to match the application execution requirements to the smallest available core that can achieve the execution within required performance limitations with lowest power consumption. In other cases, one big processor is utilized for controlling the scheduler of multiple smaller cores [10]. Previous works have suggested that this combination can reduce power consumption by 39% [11].

CPU-GPU Combinations: These platforms aim to accelerate graphics performance by including specialized hardware to execute graphics functions. The Graphics Processor Units (GPUs) consist of multiple execution units and local memory that can efficiently process data and task parallel instruction streams at a very high throughput. This architectural property also make GPUs good candidates for executing non-graphical functions that can exploit the data and task parallelism at a large scale leading to evolution of a rich General-Purpose computing on Graphics Processing Units (GPGPU) based offloading frameworks.

Hardware Accelerators: Increasingly specialized hardware accelerators are being integrated on different platforms to provide faster execution times for domain-specific application bottleneck functions. The accelerators range from fixed function units to customizable and programmable hardware (e.g. FPGAs). In conjunction with the processor cores, the specialized hardware accelerators assist in making the targeted applications perform better by taking care of the functions that use a majority of core cycles. This model makes accelerator integration a natural choice for platforms that target specific usages and/or operate under real-time and power constraints.

Remote Offloading: One of the contributions we argue in this paper is to extend the definition of heterogeneity to include a new kind of platform component – the execution units on remote resources. With the advent of mobile era, remote execution to remote resources has been one of the most common solutions to achieve better application performance. Due to the compute, memory and power limitations on mobile devices, most of the applications divide their execution between the portions of local processing and remote execution handling much of the complex and heavy task. We aim to extend this model by providing mobile applications with the flexibility

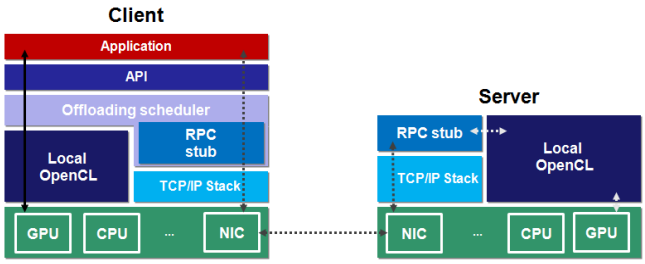


Fig. 1: System architecture.

of using remote resources through a well-defined framework, namely OpenCL.

B. OpenCL

Currently, OpenCL (Open Compute Language) is the most well-known framework for offloading tasks to different compute units within a heterogeneous host platform. This framework provides an APIs which make it possible for application developers to dispatch kernels for execution. It is supported by the top chip makers such as Intel, Nvidia, AMD and ARM Holdings. At the moment, OpenCL is used in the scientific computing arena as means to leverage GPUs for high-throughput computations. We implement our design as a subsystem of the OpenCL framework. Our extension makes it possible for the OpenCL framework to discover accelerators located on remote resources across the network. Once these remote OpenCL devices are discovered, the application developer is able to use the same interface to transparently offload a kernel as if it was part of the local platform.

III. Offloading framework

The overall architecture of the offloading framework consists of an integration with the OpenCL API, a RPC-based offloading mechanism, a dynamic scheduler. Figure 1 depicts the overall architecture of our design.

A. OpenCL API Extension

Since OpenCL is a hardware-level offloading framework and an open standard for parallel computing to improve the application performance currently supported by most of the major device manufacturer, we decided to implement a workload offloading mechanism by extending OpenCL API to support offloading over the network while the framework inherits the ability of parallel computing of the OpenCL standard. The OpenCL API possesses a set of following features: device and enumeration, device selection and customization, buffer management, job offload and status queries. With these features, the application developer has full control on the use of specific accelerators necessary to optimize their application performance. In this subsection, we explain how we extend each functionality to support remote offloading. Table I summaries OpenCL API functionalities and extensions.

Discovery and Enumeration: In the initialization phase of the interface, the developer queries the platform for on-board OpenCL-capable accelerators. The OpenCL framework returns a list of accessible compute devices located on-board the device along with their capabilities (e.g. graphics cards, video decoders, cryptographic devices). In the case of a mobile device, this might include a GPU, or a specialized stream processor [12]. We extend this portion of the API by allowing the developer to discover other OpenCL-enabled accelerators located on remote resources over the network. Hence, when the developer performs this query on a mobile device, it may discover not only the mobile GPU but also another GPU running on the remote resources.

Selection and Configuration: In the standard OpenCL framework, once presented with a list of devices, the developer selects one or more targets for workload offloading. This selection is usually based on the characteristics of each particular device (e.g. number of compute units of the accelerator, maximum number of work items, architecture, latency and so on). Based on the characteristics the developer is also able to configure the workload appropriately for better performance. Since this is a hardware layer offloading mechanism, the OpenCL framework handles support for different architectures by using a compiler which converts the code from a modified version of C99 to the target instruction set of the accelerator. Hence, the API also provides access to compiler-based customizations for both local and remote devices. By extending the discovery process to include remote OpenCL devices over the network, this selection and configuration process can become cumbersome to the developer. Thus, our system can present just one virtual device handle which represents the best offloading resource according to network conditions. Our analysis shows that bandwidth and latency have the most impact on the performance.

Workload State Transfer: Having selected a device, the next phase is the actual offloading of the data and code necessary to run remotely. The function to be executed (called a kernel) is first sent either as C99 source code or an LLVM-based intermediate language. Once transferred, the code is compiled for the target accelerator. In order to execute the kernel on the accelerator, the necessary state has to be transferred to the device regardless of whether it is local or remote. If the device resides on the host platform, the task of buffer management simply involves copying data from main memory to local storage accessible by the accelerator. However, if the workload is being offloaded to a remote resource, then the buffers have to be managed differently. First, the data has to be marshaled and copied into the networking stack's buffers then transported over the network to the appropriate remote host. The data is then copied from the remote host's networking stack into the accelerator's own local storage. For example, in the case of a mobile device offloading workloads to a GPU hosted in the remote resource, the necessary input and output buffers have to be created and copied in the GPU's local memory in the remote resources. Upon completion, the output buffers are copied back from the GPU's local memory to the mobile device's memory over the network. Also, it is possible to perform data compression at the networking layer to minimize the networking overhead and energy consumption.

Resource and Failure Management: The final phase of the OpenCL API is the ability to discover errors and release its state and resources in a graceful manner. Each function has its error parameter which keeps the developer aware of the proper execution of the remote job. If an error occurs due to an issue with the source code, or the workload configuration, or any other hardware issues, an appropriate error code is returned to the developer. In return, the developer can release

TABLE I: OpenCL API Functionalities and Extensions

Feature	API Function	OpenCL Feature	Extension Capabilities	Future Capabilities
Discovery and Enumeration	*clGetPlatformIDs*, *clGetDeviceIDs*, *clGetDeviceInfo*	List on-board devices	List the best remote resource	List all remote resources
Selection and Configuration	*clCreateContext*, *clBuildProgram*, *clCreateKernel*, *clSetKernelArgs*, *clEnqueueNDRangeKernel*	Enables job compilation and configuration	Configuration for the remote resource	Configure multiple resources
Workload State Transfer	*clCreateProgramWithSource*, *clCreateBuffer*, *clEnqueueReadBuffer*, *clEnqueueWriteBuffer*	Source code and data transfer over the internal	Source code and data transfer over the network	Explore caching for better performance
Resource and Failure Management	*clWaitEvents*, *clFlush*, *clFinish*, *clReleaseBuffer*, *clReleaseProgram*	Check on status and cleanup	Add support for network failure and cleanup	Enable timeout for slow responses

the various resources (i.e. buffers, device handles) that are associated with the job. We extend this functionality to support network failures. In case of a disconnection, the appropriate error code is returned to the developer who then performs the necessary actions to clean up the state belonging to the job. On the server, the necessary clean-up is taken as well by our framework. Our decision utilize the OpenCL framework for workload offloading in mobile devices allows us to leverage all of the functionalities already in place for offloading workload locally from the GPU to an on-board hardware accelerator. We added the required extensions to the framework by creating a wrapper library around the OpenCL API. Since we provide an identical interface, developers can integrate their OpenCL code with our system without an code modification. In the next subsection, we describe how we integrate the OpenCL API with RPC to support remote workload offloading.

B. Integration of OpenCL API with RPC

The integration of OpenCL API with RPC-based service is realized as a wrapper library which provides an identical interface to the original OpenCL API. However, the offloading framework intercepts each API call in the application layer into the wrapper library. In the wrapper library, if a scheduler (a runtime scheduler is beyond this work) decides to offload the workload, it routes these API invocations to the appropriate remote resource through RPC, otherwise, it invokes corresponding OpenCL API locally.

In our first attempt, we utilized SunRPC to provide the remote procedure call interface, serialization, and networking capabilities. However, SunRPC provides many extra features that were not necessarily efficient, for example, the use of a portmapper daemon to discover the listening port of the RPC service. SunRPC also initiates a new TCP connection for each function call which incurs extra costs. In contrast, our RPC-based service uses a single TCP connection per offloading job achieving better performance than SunRPC. By running an RPC service which exposes the OpenCL API over the network, we provide the workload offloading design that is lightweight in terms of argument serialization and buffer management.

IV. Methodology

The main contribution of this work is to analyze the behavior of the remote offloading in accordance with the characteristics of the workloads and environmental factors such as network conditions and the level of computing capabilities of remote resources. In this section, we explain the workloads used for the analysis and the way to characterize the workloads as well as experimental setup. Every experiment is repeated 5 times and the results presented in the paper are the averaged values.

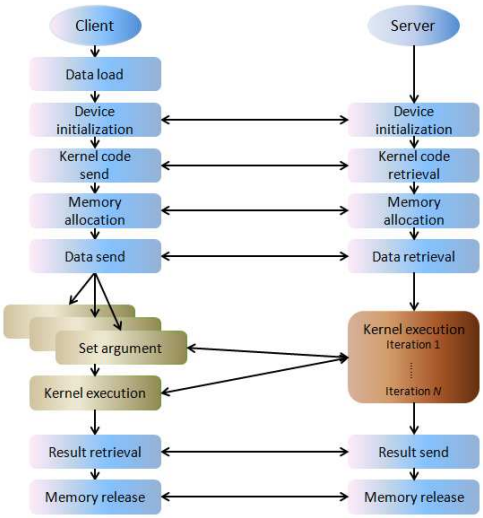
TABLE II: Average and standard deviation of network latency and bandwidth for local and wide area networks including Amazon EC2.

	LAN		Campus network		Amazon EC2	
Latency (*ms*)	Avg. 10.833	Stdev. 2.684	Avg. 15.465	Stdev. 4.189	Avg. 74.036	Stdev. 17.737
Bandwidth (*MB*)	Avg. 6.523	Stdev. 0.177	Avg. 2.461	Stdev. 0.238	Avg. 0.178	Stdev. 0.023

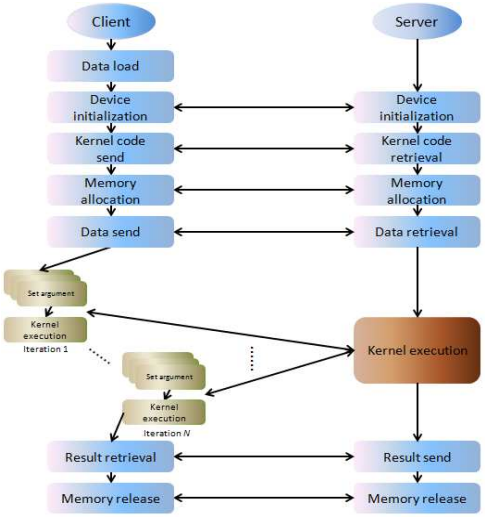
A. Hardware setup

In order to analyze the behavior of our remote offloading framework under the various network conditions and the computing capabilities of remote resources, we setup the experiment using various hardwares and network configurations. First of all, our hardware setup consists of a client and various types of remote resources. We have utilized Android tabletPC, Samsung GalaxyTab 10.1 equipped with 1GHz dual-core process and 1GB RAM, and running Android honeycomb as the mobile client. We adopted three types of servers: CPU only-installed server, GPU-installed server, and Amazon EC2 GPU cluster. For CPU only-installed server without a graphics card, we utilized a workstation with Intel 3.0GHz Core 2 Duo processor and 8GB memory installed with Linux OS. Also, we installed GeForce GT 640 graphics card with 2GB RAM for the server with GPUs. In addition, we used Amazon EC2 GPU cluster resource with two Nvidia GPUs equipped with 3GB on-board memory per GPU.

In order to analyze the impact of network conditions on offloading performance, we configured both local and wide area networks using following setup. Firstly, we built a local area network by directly connecting the mobile client and the server through a wireless router supporting 802.11 b/g/n network standard. Secondly, we used our campus network in order to configure a wide-area network in which the client and the server connect to a wireless and wired network



(a)



(b)

Fig. 2: Program flow diagram for OpenCL workloads

respectively. Also, we emulate different wide-area network conditions between the client and the server by controlling the network latency using Traffic Control (TC) [13]. TC is a network tool which provides functionalities to control network traffic by priotizing network resources and using concepts of traffic classification, queue disciplines and quality of service(QoS). Furthermore, utilizing an instance of Amazon EC2 GPU cluster gives us one more option for the server located in wide-area network. Table II shows the average and standard deviation of network latency and bandwidth for our network configurations.

B. OpenCL workloads

We utilized OpenCL SDK code samples provided by AMD APP SDK [14] and Nvidia [10] to choose appropriate workloads for the remote offloading framework. In order to choose appropriate OpenCL workloads among a set of samples, we have reflected the following simple, yet definite expectation into the choice of the workloads.

"The more computation- and less communication-intensive the workload, the more potential gain from remote offloading"

Furthermore, we characterized the workloads in accordance with the amount of computation and communication for input and output data transfer by considering computation to communication ratio calculated by the execution time required to process a certain amount of data locally divided by the data transfer time for the remote offloading. Thus, computation to communication ratio is a comprehensive measurement which mirrors three parameters such as the the volume of computation of workloads, the amount of data to be transferred, and the network conditions. After careful considerations, we chose four qualitatively and quantitatively different workloads: Sobelfilter, matrix multiplication, hidden Markov model, and *N*-body physics used by a variety of areas such as image processing, physics simulation, and mathematical modeling. Also, since it is necessary to consider an additional interaction between the client and the server such as device configuration or state tranfer which occurs further overhead for communication, we examined the programming flow of the application-layer for each workload. The programming flow and structure of workloads are described in the following:

Sobelfilter: Sobelfilter is an image processing filter used for image edge detection. It consists of both a 3×3 horizontal and vertical filter. The edge detection process applies two filters into an input image in a sequence and adds final results to a form of the image. Accordingly, the client transfers the horizontal, vertical filter, and the input image as states for the execution and sets them as arguments accessed by the kernel. In the server side, the image is iteratively processed based upon the filters that the client has sent. Once the execution is completed, the server sends back an output image into the client. Figure 2(a) shows the program flow diagram for the data transfer, the argument setup, and the kernel execution for Sobelfilter.

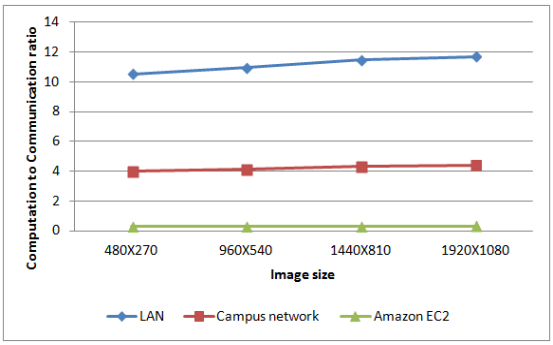
Matrix Multiplication: Similarly to Sobelfilter, the client sends to floating-point matrixes (*n* by *n* and *n* by *2n*) and additional small states and arguments necessary for the remote kernel execution. Then, server executes multiplication of two matrices and returns one *n* by *2n* matrix to the client as the result. Therefore, matrix multiplication also follows the program flow of Figure 2(a).

***N*-body physics:** *N*-body physics is a mathematical simulation for predicting a dynamic system consisting of particles or astronomical objects under physical forces such as gravity. In our OpenCL workload for *N*-body physics, the client generates initial states for a certain number of objects and sends them as an input data. However, the kernel takes additional arguments for each iteration of kernel execution incurring additional communication between the client and the server for the arguments setup. The program flow for *N*-body physics is shown in Figure 2(b).

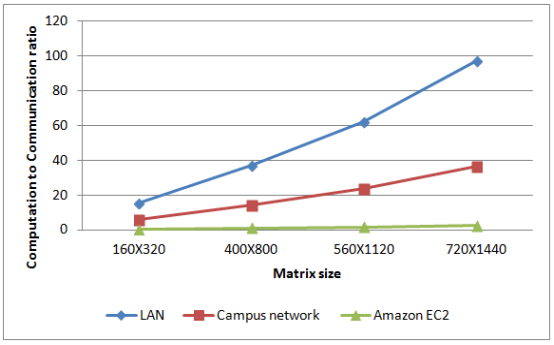
Hidden Markov model: Hidden Markov model is a popular statistical tool for modeling generative sequences which can be characterized by observable sequences. It has been applied to a wide range of applications such as image, speech recognition, computational biology, bioinformatics and

TABLE III: Summary of input, output data, and the number of Argset API calls(total data for Argset API calls). The unit is KBytes

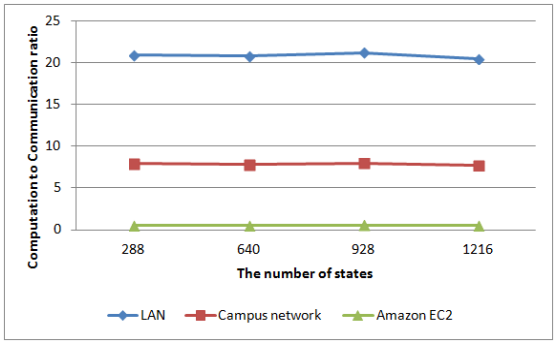
Sobelfilter					Matrix multiplication				
Image size	480×270	960×540	1440×810	1920×1080	Matrix size	160×320	400×800	560 ×1120	720×1440
Input	390	1,520	3,310	5,790	Input	270	1,720	3,390	4,610
Output	110	370	720	1,190	Output	170	1,280	2,010	3,340
Number of Argset calls	7(0.5)	7(0.5)	7(0.5)	7(0.5)	Number of Argset calls	8(0.05)	8(0.05)	8(0.05)	8(0.05)
Hidden Markov model					*N*-body physics				
Number of states	288	640	928	1,216	Number of iterations	5	10	50	100
Input	290	1,470	3,200	5,500	Input	15	15	15	15
Output	78	108	110	119	Output	30	30	30	30
Number of Argset calls	1000(66)	1000(66)	1000(66)	1000(66)	Number of Argset calls	20(0.16)	40(0.32)	200(1.6)	400(3.2)



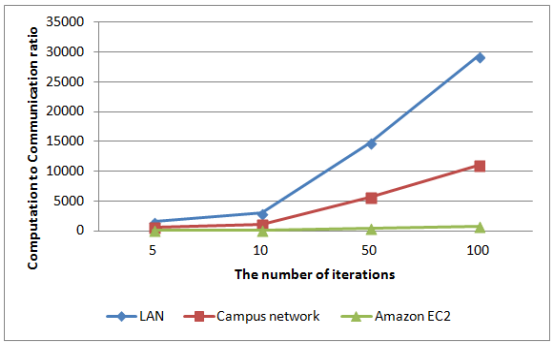
(a) Sobelfilter



(b) Matrix multiplication



(c) Hidden Markov model



(d) *N*-body physics

Fig. 3: Computation to communication ratios of four workloads for the various data size and network configurations

environment engineering. The OpenCL workload for hidden Markov model follows similar steps for kernel execution as *N*-body physics. The client generates and transfers initial states and transition probabilities to the server. Also, the client sends different arguments in order to execute each iteration of the kernel. Therefore, the program flow for hidden Markov model is represented by Figure 2(b).

V. Analysis

In this section, we analyze the behavior of the OpenCL-based remote offloading framework for mobile platforms in accordance with the workload characteristics and environmental factors such as network conditions and the computing capabilities of remote resources through real deployment over local and wide area environments. First of all, we observe computation to communication ratios of the workloads for the given size of data and network configurations. Next, we evaluate the efficacy and the cost of the remote offloading framework with the various network configurations and the remote resources. Table III summarized the amount of input and output data transfer, and the number of additional API calls for argument setup required to execute the OpenCL kernel code for the given size of data or the number of iterations.

A. Computation to communication ratio

In this section, we present computation to communication ratio for each workload for the given size of data and network configurations. As we expected, it is evident that for all workloads, computation to communication ratio with local area networks is higher than other network configurations, since the data transfer time becomes shorter by higher network bandwidth from local area networks. On the contrary to local area networks, the cases of Amazon EC2 have the lowest computation to communication ratio because of its most restricted

network conditions. As shown in Figure 3(a) and (c), however, computation to communication ratios for Sobelfilter and hidden Markov model are relatively uniform regardless of the size of data, while those for matrix multiplication and *N*-body physics increase as the size of data or the number of iterations increases. This is because that the volume of computation for Sobelfilter and hidden Markov model is linearly proportional to the amount of data to be processed, while that for matrix multiplication and *N*-body physics increases exponentially as the size of data or the number of iteration increases, which means that the benefits from offloading such workloads to more powerful resources also increase exponentially. Therefore, we expect it is highly likely that more gain is available from offloading matrix multiplication and *N*-body physics as the size of data increases,

B. Offloading performance

First of all, for Sobelfilter, we observed better performance from only a few cases where offloading 1440×810 and 1920×1080 of the image size to remote resources located in local area network than local processing in Figure 4(a). In fact, Sobelfilter has lower computation to communication ratios that other workloads ranging from 0.28 to 11.68 in our experimental setup which indicates it is less computation-intensive (i.e. more communication-intensive) workload. Especially, the cases of offloading to the resources located in campus network and Amazon EC2 cluster instance whose computation to communication ratio is fairly low have the worst performance in Sobelfilter. Also, we observed that better performance comes from more powerful computing power by comparing CPU only-installed server and GPU installed server in the same network configuration. In contrast to Sobelfilter, in all the cases of matrix multiplication except for 160×320 of matrix size, offloading is much faster than local processing showing the speed-up which ranges from 1.2× to 9.2× as shown in Figure 4(b). As mentioned in previous section, computation to communication ratios of matrix multiplication with our setup range from 0.41 to 97.42 which means that matrix multiplication is more computation-intensive workload than Sobelfilter. Therefore, it is highly likely that matrix multiplication is able to gain more from offloading to the remote resources with more powerful computing capabilities. In fact, the computation for matrix multiplication has higher complexity than Sobelfilter (the computation for matrix multiplication is $O(n^3)$ while Sobelfilter is $O(n^2)$). For the smallest matrix size of our setup, 160×320, however, we observed similar results as 1440×810 and 1920×1080 of image size in Sobelfilter where only offloading to resources in local area networks has better performance than local processing. With the results of Sobelfilter and matrix multiplication, it is evident that as computation to communication ratio becomes higher, it is possible to gain more from offloading.

Interestingly, in the cases of hidden Markov model, the extremely worst performance is shown when the workload is offloaded to Amazon EC2 GPU cluster which has the worst restrictions in terms of network latency and bandwidth. Though hidden Markov model has higher computation to communication ratio than Sobelfilter, the kernel for hidden Markov model is repeatedly executed requiring additional argument setups for each execution which incur more communication overhead as described in Section IV.B. In the Amazon EC2 setup, packets are sent or received at higher latencies compared to the local area networks which impacts performance especially since our offloading framework requires that each RPC call is acknowledged with a response from the server. Consequently, offloading to Amazon EC2 GPU cluster, which has the highest latency among our experimental setups, takes the longest time. However, though *N*-body physics has the similar programming flow as hidden Markov model, all the cases of offloading have better performance than local processing because it is the most computation-intensive workload among our experimental setup whose computation to communication ratios range from 40.36 to 29323.68.

C. Energy consumption

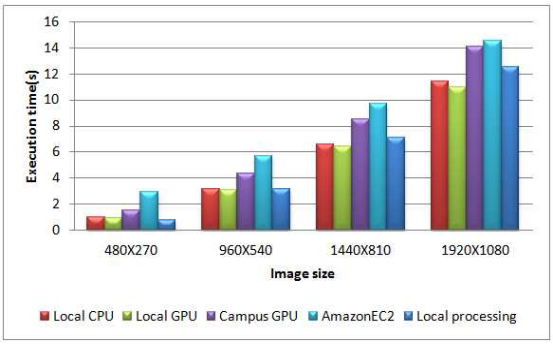
To profile energy consumption of the mobile device, we used PowerTutor [15] which is an application for the variants of Android devices that displays the power consumed by major components such as GPU, network interfaces, LCD display, and GPS receiver.

A similar pattern for energy consumption of the mobile device as the execution time is shown in Figure 5. As computation to communication value is higher, it is likely that offloading is more beneficial than local processing. However, it is worth noting that though some cases for Sobelfilter showed the benefits from offloading in terms of the execution time, offloading consumes more than local processing as shown in Figure 3(a). This dissimilar result comes from the discrepancy in the amount of power consumed by CPU and the Wi-Fi networking card. According to our measurement data profiled by PowerTutor, CPU consumes 200∼220*mW* per second in active mode, while the Wi-Fi networking card consumes about 710∼720*mW* per second in high power mode. For that reason, even though offloading is faster than local processing, it is possible that offloading consumes more energy than local processing. However, in matrix multiplication and *N*-body physics which result in extremely high speed-up by offloading, it is observed that offloading also improves energy implication as shown in Figure 5(b) and (d).

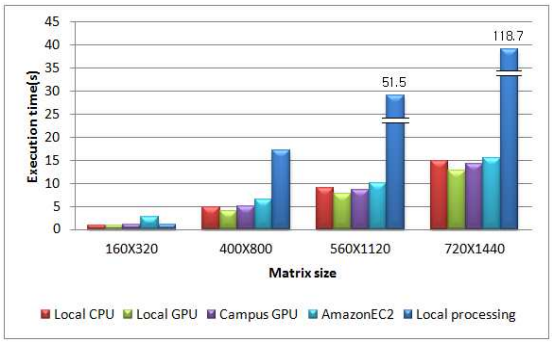
VI. Related work

A. Mobile workload analysis

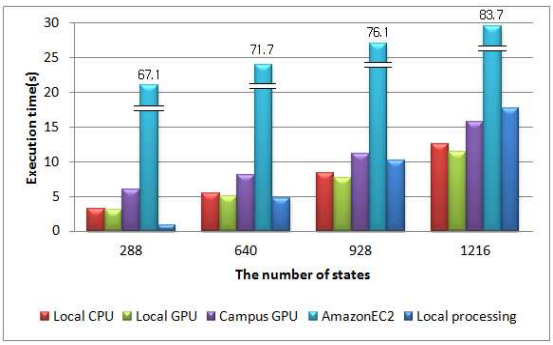
In [16], the authors characterize the microarchitectural behavior of smartphone applications through several representative benchmarks including the following areas: an interactive game, a streaming player and mp3 audio player. Also, they developed a new benchmark to characterize the performance of a web browser called BBench. Through those benchmarks, they show how they differ from SPEC CPU2006 benchmarks which are widely used for the measurement of compute-intensive performance. MEVBench [17] presented a custom benchmark suite for full mobile vision applications such as augmented reality as well as components of common vision algorithms such as SIFT feature extraction and SVM classification. The authors evaluated the performance of MEVBench with various platforms for the direction of future mobile embedded vision architectures. Also, they show that mobile vision architectures require to be supported from heterogeneous computation for performance improvement. In [18], the performance and energy benefits of mobile heterogeneous computing are characterized by using 2D FFT(Fast Fourier



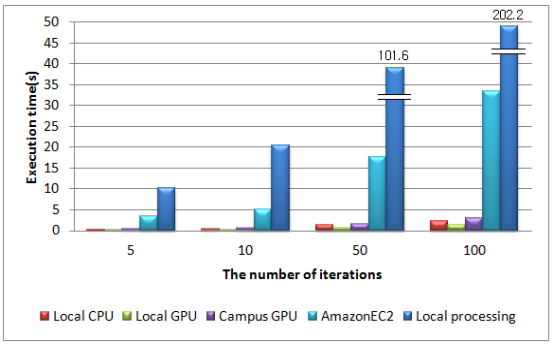
(a) Sobelfilter



(b) Matrix multiplication

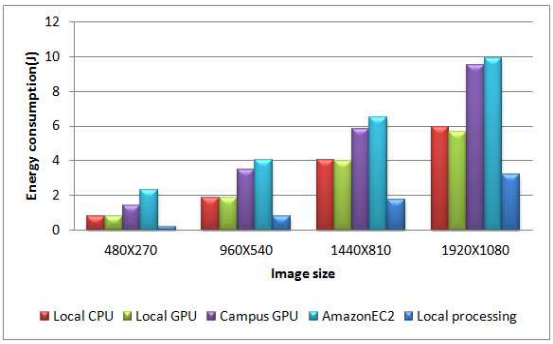


(c) Hidden Markov mode

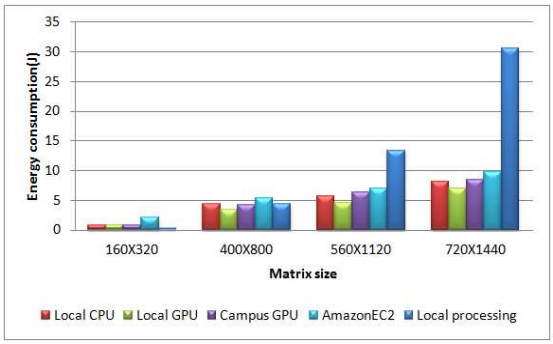


(d) *N*-body physics

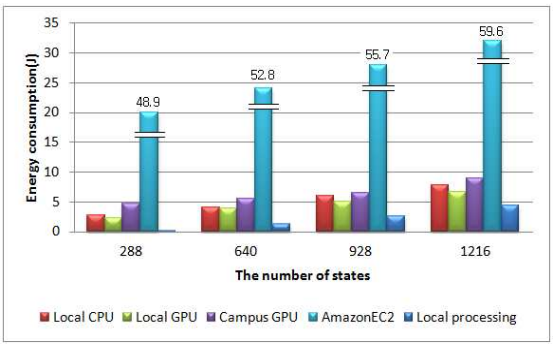
Fig. 4: Comparison of total execution time for four OpenCL workloads with various servers and network setup



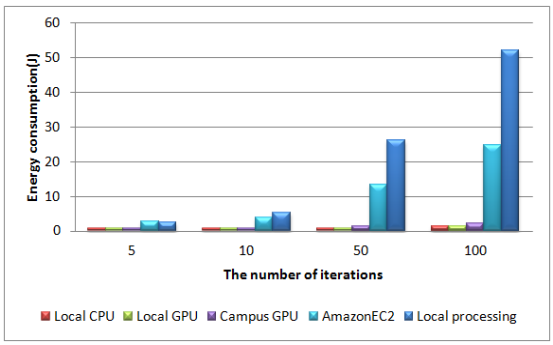
(a) Sobelfilter



(b) Matrix multiplication



(c) Hidden Markov mode



(d) *N*-body physics

Fig. 5: Comparison of energy consumption for four OpenCL workloads with various servers and network setup

Transformation), matrix multiplication, and 2D Stencil as the benchmarks. In this study, the authors demonstrated that fully utilizing available computing cores to complete a task can achieve an 3.7× speed-up over the case of the single-threaded CPU, consequently, reduce the energy consumption for the mobile platform.

In contrast with the aforementioned studies which designed new benchmarks or characterized the performance and energy benefits of the mobile platforms via the various workloads from a perspective of mobile computing capabilities, we characterized the behavior of mobile offloading framework while considering the network conditions as well as the computing capabilities of remote resources.

B. Remote offloading framework

The research community has proposed different methods to offload the computation for the mobile platforms as well as high performance clusters. In Spectra [19], developers identify functions in the application that can be offloaded to a remote server over RPC. By monitoring the CPU, file system, and bandwidth, Spectra dynamically decides at run-time which portions of the application should run locally or remotely. MAUI [20] takes a similar approach but alleviates the process by using many of the programming features in the .NET platform such as method attributes, and the Reflection API. Through the .NET framework's virtual machine, MAUI is able to dynamically serialize and ship remotable methods and data to a server proxy. Cuckoo [21] takes a slightly modified approach by focusing more on integrating with the Eclipse IDE, however it requires developers to implement both local and remote versions of their functionality, whereas MAUI only requires annotations instead of a different implementation. CloneCloud [22] achieves the remote offloading by employing thread migration in the Dalvik Java virtual machine (JVM) by transferring all of the thread state (thread stack, necessary heap objects and registers) to the remote virtual machine. When the remote thread completes, the results are merged back with the local Dalvik JVM memory stack. There also exist other heterogeneous offloading frameworks which are similar to our approach. Remote CUDA [7] is one approach that extends the Nvidia CUDA API to support remote offloading over the network. Their goal is to minimize the network overhead and they analyze the impact of using different networking technologies such as Gigabit Ethernet or InfiniBand. Virtual OpenCL [8] provides a similar solution which uses OpenCL instead of the proprietary CUDA protocol. They also focus on cluster environments but they leverage the MPI library for memory and workload synchronization.

Although we are not the first to propose the concept of offloading heterogeneous workloads over the network, our design is the first to consider this approach where offloading happens into mobile platforms. Furthermore since our framework is designed for mobile platforms, it differs greatly because it takes into account not only performance, but also power energy consumption.

VII. Conclusion

In this paper, we analyzed the behavior of the mobile offloading framework in terms of the offloading performance and energy implication of the mobile devices with respect to characteristics of workloads and environmental factors such as network conditions or the computing capabilities of remote resources. In order to characterize the workloads, we used computation to communication ratio calculated by the local processing time divided by the time for data transfer. Furthermore, as part of the workload characterization effort, we developed the OpenCL-based remote offloading framework by broadening the scope of heterogeneity to include a new kind of platform component: the computing capabilities on remote resources. We accomplish this by extending the well-defined hardware-level offloading standard, OpenCL framework to support the various range of remote computing elements over the network using the regular TCP/IP network stack. Also, we configured both local and wide area networks in which the various computing capabilities are deployed to evaluate the impact of environmental factors into the offloading performance. According to the analysis, the benefits and the costs of the remote offloading depend on computation to communication ratio as well as the programming flow related to how the client and the server interact each other. In fact, as computation to communication ratio becomes higher, the performance improvement and the conservation of energy consumption also increase. Moreover, in the cases of hidden Markov model and N-body physics, we observed that the programming flow is also important to analyze the behavior of the remote offloading for the mobile platforms. We believe that the numerical results presented in this paper guides the basis of designing the intelligent runtime offloading scheduler, so we are currently working on a machine learning-based runtime offloading scheduler. Also, we are currently seeking to real mobile applications such as face and voice recognition, or mobile gaming so that we are able to evaluate the proposed offloading framework with those applications.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 0758596. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] L. Dignan, "Android cumulative shipments to hit 1 billion mark in 2013." sep 2012.

[2] S. Chen, "An introduction to heterogeneous optimal selection theory." in *TENCON*, vol. 1. IEEE, 1993, pp. 138–141.

[3] "Intel corporations." [Online]. Available: http://www.intel.com

[4] "Advanced micro devices incorporation." [Online]. Available: http://www.amd.com

[5] H. Eom, P. S. Juste, R. Figueiredo, O. Tickoo, R. Illikkal, and R. Iyer, "Snarf: A social networking-inspired accelerator remoting framework." in *In proceeding of Workshop on Mobile Cloud Computing(MCC)*. ACM, 2012, pp. 29–34.

[6] M. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing." in *In proceeding of IEEE International Conference on Computer Communications(Infocom)*, 2013.

[7] J. Duato, A. J. Peña, F. Silla, R. Mayo, and E. S. Quintana-Ortí, "rcuda: Reducing the number of gpu-based accelerators in high performance clusters." in *In proceeding of International Conference on High Performance Computing and Simulation(HPCS)*, 2010, pp. 224–231.

[8] S. Xiao, P. Balaji, J. Dinan, Q. Zhu, R. Thakur, S. Coghlan, H. Lin, G. Wen, J. Hong, and W. chun Feng, "Transparent accelerator migration in a virtualized gpu environment." in *In proceeding of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGRID)*. IEEE/ACM, 2012, pp. 124–131.

[9] "Intel® atom processor." [Online]. Available: http://www.intel.com/content/www/us/en/processors/atom/atom-processor.html

[10] "Nvidia opencl sdk code samples." [Online]. Available: http://developer.nvidia.com/opencl

[11] R. Kumar, K. I. FarKas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction." in *In proceeding of IEEE Workshop on Signal Processing Systems(SiPS)*, 2012, pp. 312–317.

[12] "Leverage stemcell compute performance with opencl." [Online]. Available: http://www.ziilabs.com/products/software/opencl.php

[13] W. Almesberger, "Linux network traffic control – implementation overview." 1999.

[14] "Accelerated parallel processing(app) sdk with opencl 1.2 support." [Online]. Available: http://developer.amd.com/tools/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/

[15] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones." in *In proceeding of International Conference on Hardware/Software Codesign and System Synthesis(CODES+ISSS)*. ACM, 2010, pp. 105–114.

[16] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. N. Mudge, A. G. Saidi, C. Emmons, and N. C. Paver, "Full-system analysis and characterization of interactive smartphone applications." in *In proceesing of IEEE International Symposium on workload Characterization(IISWC)*. IEEE, 2011, pp. 81–90.

[17] J. Clemons, H. Zhu, S. Savarese, and T. M. Austin, "Mevbench: A mobile computer vision benchmarking suite." in *In proceeding of IEEE International Symposium on Workload Characterization(IISWC)*. IEEE, 2011, pp. 91–102.

[18] Y. C. Wang and K. T. T. Cheng, "Energy and performance characterization of mobile heterogeneous computing." in *In proceeding of IEEE Workshop on Signal Processing Systems(SiPS)*, 2012, pp. 312–317.

[19] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing." in *In proceeding of International Conference on Distributed Computing Systems(ICDCS)*, 2002, pp. 217–226.

[20] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Ch, and P. Bahl, "Maui: making smartphones last longer with code offload." in *In proceeding of International Conference on Mobile Systems, Applications and Services(MobiSys)*. ACM, 2010, pp. 49–62.

[21] R. Kemp, N. Palmer, T. Kielmann, and H. E. Bal, "Cuckoo: A computation offloading framework for smartphones." in *In proceeding of International Conference on Mobile Computing, Applications and Services(MobiCASE)*, vol. 76. Springer, 2010, pp. 59–79.

[22] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud." in *In proceeding of the European Conference on Computer Systems(EuroSys)*, 2011, pp. 301–314.