A Semi-Automated Tool Flow for Roofline Anaylsis of OpenCL Kernels on Accelerators

Servesh Muralidharan
ICHEC
Dublin, Ireland
servesh@ichec.ie

Kenneth O'Brien
Xilinx Inc
Dublin, Ireland
kennetho@xilinx.com

Christian Lalanne
ICHEC
Dublin, Ireland
clalanne@ichec.ie

ABSTRACT

We propose a tool-flow methodology that can be applied to analyze and track the performance of OpenCL applications on heterogeneous platforms. Using a case study on a datacenter representative workload, we evaluate our tool flow on three distinct heterogeneous platforms and demonstrate how it can be employed more widely to provide insight and track attainable performance of OpenCL applications. Our methodology is motivated by the need for a common set of metrics that can characterize the performance and power efficiency of OpenCL applications on the increasingly diverse range of emerging heterogeneous platforms now relevant to both HPC and the datacenter market.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Measurement

Keywords

Roofline, OpenCL

1. INTRODUCTION

At a time when Von Neumann architectures are suffering from rising power densities and when it is now generally accepted that leveraging multi-core CPUs alone is no longer a power-efficient approach to supercomputer and datacenter design, extreme-scale computing communities are turning to platforms with increasing heterogeneity as a solution to power constraints. Such heterogeneous platforms are typically composed of traditional multi-core CPU processors combined with many-core accelerators or co-processors, which typically offer higher performance per Watt than conventional CPUs for highly parallel workloads. Currently, the two most widely exploited many-core platforms in the Top500 list of supercomputers [22] are general purpose graphics processing units (GPGPUs) and the Intel Xeon Phi Many Integrated Core (MIC) co-processor. At the same time, web service and social media companies, with huge demands to solve problems in speech recognition, image search and natural language processing are determining that, for many such workloads, FPGA-based solutions scale far more efficiently than CPUs while also reducing power [18].

In this paper, we have been particularly stimulated by the recent widening of OpenCL support to include energy efficient FPGA-based platforms [24, 5]. Such platforms are becoming increasingly relevant to certain types of HPC-like workload in the datacenter [18], and may increase in relevance to the HPC community as we face into the challenging demands placed on energy efficient exascale computing by 2020. We have also been motivated by the increasing availability of OpenCL supported tools as well as the growing momentum building behind the Standard Portable Intermediate Representation (SPIR) language for parallel computing by the Khronos Group, which enables the creation and distribution of device-independent binaries within the OpenCL stack [12]. With these initiatives and challenges in mind, we propose here a common set of high-level OpenCL-centric figures of merit, along with a tool-flow methodology that leverages the well-known roofline model [23] for comparing performance and power across a wide range of heterogeneous platforms, and which extends its reach to more novel FPGA-based platforms for the first time. Also, we do not consider the host system in our tool-flow but only measure and compare the performance of the accelerators. The main contributions of this paper are:

- A proposed common set of performance-related metrics relevant to OpenCL applications and platforms in section 3
- A proposed semi-automated OpenCL-centric tool flow methodology for measuring and comparing performance and power in section 4
- An initial evaluation of the proposed methodology using a non-conventional workload in section 5

2. MOTIVATION

Comparing a diverse set of OpenCL supported platforms on a common set of metrics is a non-trivial problem which comes with the challenges of dealing with different instruction set architectures (ISAs), memory hierarchies, cache models and hardware counters as well as the availability (or lack) of common profiling tools. The widening of OpenCL supported platforms to those based on reconfigurable logic or FPGAs increases this challenge even more so. Furthermore, benchmarking a device based on a single OpenCL kernel has

its limitations, as optimizations performed on one platform may or may not lead to optimal performance on a different platform. An example of this would be GPUs which perform well with OpenCL kernels that exploit data parallelism. If the same kernel were to be optimized to target task parallelism it might not achieve optimal performance. In such a scenario not only does it incorrectly show that a particular accelerator performs poorly, it also gives an inaccurate impression of the application's compatibility for the specific hardware.

In posing the question as to what is a fair comparison across architectures and OpenCL kernel designs, we feel that the best starting point is the well established roofline model [23]. The roofline model combines the application's performance and the device's capability in a single diagram. This is achieved by plotting the performance of executed code against its operational intensity. It also includes two platform-specific performance ceilings: the accelerator's peak performance and a ceiling derived from its memory bandwidth. The model can thus differentiate between applications and optimizations that are memory or compute-bound on the given architecture. Moreover, we can determine if we have achieved the peak performance or if further optimizations can be performed as well as possibly obtaining insight into the suitability of an accelerator for the given application.

While it has already been shown that rigorous application of the roofline model can aid in tracking attainable performance on a given device [16, 4], the approach has to date been quite limited in the diversity of platforms that it has been employed on. Therefore, we propose to employ and extend roofline models to OpenCL kernels and accelerators, including FPGA-based platforms for the first time. In section 3 we define a set of OpenCL-centric metrics that is used later in our semi-automated tool flow methodology.

3. PRELIMINARIES

Inspired by the methodologies found in [16] and [4], we first set out to clearly define a set of metrics that can be applied to the wide range of hardware platforms that we initially consider as part of this work. These metrics form the basis of quantifying several measurements that we perform and later use in our methodology. The fundamental metrics defined in this section are those which can be measured directly from an experiment using a specific tool. The derived metrics are obtained from these using mathematical formulations taken from the roofline model.

3.1 Fundamental Metrics

In the rest of this paper, we use the traditional terminology in the field of heterogeneous computing. By CPU, processor or "host" system, we generally mean a CPU-based computing system, such as an x86 Xeon processor. By accelerator or coprocessor, we mean a manycore "device" in the form factor of a PCIe card. Also by global memory we mean the DRAM memory on the PCIe card and local memory corresponds to the on-chip memory of the coprocessor.

Device Memory Bandwidth ($\mathcal{B}$) : denotes the memory bandwidth between the global memory and local memory on an OpenCL device as specified in the datasheet of the device. **Device Peak Memory Bandwidth** ($\hat{\mathcal{B}}$) considers only the peak bandwidth that can be achieved by a benchmark kernel that performs operations similar to that of the evaluated application. For example, if the application performs integer operations, a benchmark that closely reflects these operations is used. Both of these metrics are measured in 'Bytes/Second' (number of bytes per second).

OpenCL Kernel Operations ($\mathcal{W}$): is defined as the number of operations performed by a given OpenCL kernel and can refer to any type of "meaningful" operation, such as comparisons or integer arithmetic (as is the case for our initial application described in section 5). Such operations explicitly exclude operations related to data movement such as read/write or branching, etc. For the purposes of our case study investigation, $\mathcal{W}$ will count the number of compute-relative integer operations, including additions, multiplications, etc. It should be emphasised that by operations here we mean mathematical operations, so for example, one SSE addition of 2 vectors of 4 single integer values will count as 4 operations. While $\mathcal{W}$ is a property of the chosen algorithm and does not depend on the platform, there is recognition that some platforms have hardware support for non-trivial operations. In such cases, we may consider such operations in their "base" form as implemented in software; in the future we may calculate specialized $\mathcal{W}$ values for specific instruction sets. The unit of measure for $\mathcal{W}$ is 'OPS' (number of operations), which can be either float or integer based. In the case of OpenCL-based applications, we propose that the value of $\mathcal{W}$ can be determined in two ways: theoretically with static code analysis or through measurement with the use of the OCLgrind, an open source SPIR interpreter and device simulator [17] in our tool flow.

OpenCL Kernel Global Memory Traffic Size ($\mathcal{Q}$): is the number of bytes of memory traffic between the global (off-chip) memory on the device and the on-chip memory on the device incurred by executing a given application. It is measured in 'Bytes' (number of bytes). The value of $\mathcal{Q}$ depends on the properties of both the application and the platform, such as the details of the on-chip memory hierarchy and can only be estimated asymptotically in most cases with exact values determined from measurements. If all data fits in on-chip memory on the device $\mathcal{Q}$ is typically equal to the number of compulsory misses. For larger sizes of data, there will also be traffic associated with data that has been evicted from cache and potential conflict misses, but these numbers cannot be determined analytically. In our tool flow we use a combination of memory access instructions from OCLgrind and a data traffic analyzer of a particular platform to determine this number. The data traffic analyzer consists of platform specific tools such as Intel VTune [19], NVIDIA profiler [14] and Xilinx SDAccel [24].

OpenCL Kernel Execution Time ($\mathcal{T}$): The execution time for a given OpenCL kernel measured in 'Seconds', is the duration the kernel runs on a specific device. We do not take into account the time taken for loading or storing the data from the host. We use the OpenCL events profiling API to obtain this information from the runtime.

OpenCL Kernel Power Consumption ($\mathcal{P}$): Modern accelerator devices expose power consumption data through

vendor specific libraries [15, 10, 21]. We leverage these libraries to construct a power profiling tool which executes our target applications. For the lifetime of the target, the profiler records power consumption of the given device. By taking a high resolution timestamp before and after the OpenCL kernel execution, we can extract the power attributed to the kernel. Since the aforemenetioned libraries have a very low temporal resolution, we only consider the **Peak Power** consumption in 'Watt' of a kernel for our measurements. Similarly to **Accelerator Peak Memory Bandwidth** ($\hat{\mathcal{B}}$) we also define an **Accelerator Peak Power Consumption** ($\hat{\mathcal{P}}$) metric that is measured based on the same benchmark that determines the peak memory bandwidth, so as to avoid the differences in the power consumption values defined in the datasheet compared to the more realistic peak power consumption of the device.

3.2 Derived Metrics

Operational Intensity ($\mathcal{I}$): is the ratio of OpenCL Kernel Operations ($\mathcal{W}$) to the OpenCL Kernel Global Memory Traffic Size ($\mathcal{Q}$) for a given application measured in 'OPS/Bytes'.

$$\mathcal{I} = \mathcal{W}/\mathcal{Q} \tag{1}$$

Energy ($\mathcal{E}$): The energy consumed by an OpenCL kernel is determined as the product of the execution time ($\mathcal{T}$) and the peak power ($\mathcal{P}$) consumed by the kernel. The unit of measurement for this metric is 'Joule'.

$$\mathcal{E} = \mathcal{T} \times \mathcal{P} \tag{2}$$

Performance ($\mathcal{F}$): The performance of an OpenCL kernel is determined as the ratio of ($\mathcal{W}$) to execution time ($\mathcal{T}$) and is measured in 'OPS/Second'. Performance in a typical roofline model accounts for the entire system's behavior. However, so as to allow for the comparison of a diverse range of OpenCL supported accelerators with a similar experimental setup, we do not consider any influence from the host system in our measurements. Our objective is to compare and evaluate the OpenCL kernels and devices only. Therefore, measurements that are based on the host system are beyond the scope of this paper.

$$\mathcal{F} = \mathcal{W}/\mathcal{T} \tag{3}$$

Performance Per Watt ($\mathcal{R}$): The performance per Watt of an OpenCL kernel is determined as the ratio of performance ($\mathcal{F}$) to the peak power ($\mathcal{P}$) consumed by the kernel in a second and is measured in 'OPS/Second/Watt'.

$$\mathcal{R} = \mathcal{F}/\mathcal{P} \tag{4}$$

Performance per Watt is a well established parameter for comparing the energy efficiency of compute systems [8] and in this paper we propose that we can quite easily complement the traditional roofline methodology, which has typically been used to characterise performance in time to also characterise performance per Watt with very few changes to the existing model. In doing so, we note that Choi et al. [4] have put forward a more rigorous model for characterizing and comparing the energy efficiency of applications on heterogeneous platforms using the roofline methodology, where they define an energy-balance analogue to the time-balance basis of the roofline model. In future work we plan to bridge our methodology with Choi et al.'s energy roofline approach to obtain more precise energy-balance comparisons.

For a given metric we can refer to operations on different data types by subscript notations. In this paper we use '$_i$' for integers and '$_f$' for floating point data types. It should be noted that this approach can be extended to work with other types of operation, therefore making it flexible enough to apply to any kind of OpenCL kernel. We note that in some cases we need to use a benchmark or datasheet to determine its peak value as in eq.6.

$$\begin{aligned} &\mathcal{A} \in \{\mathcal{W}, \mathcal{I}, \mathcal{F}, \mathcal{R}\} \\ &\mathcal{A}_i \text{ refers to integer OPS} \\ &\mathcal{A}_f \text{ refers to floating point OPS} \end{aligned} \tag{5}$$

$$\begin{aligned} &\mathcal{M} \in \{\mathcal{B}, \mathcal{P}, \mathcal{E}, \mathcal{F}, \mathcal{R}\} \\ &\hat{\mathcal{M}} \text{ refers to benchmark value} \\ &\overline{\mathcal{M}} \text{ refers specification value} \end{aligned} \tag{6}$$

4. METHODOLOGY

The main thrust of our paper is to propose a semi-automated OpenCL-centric tool flow methodology for measuring and comparing performance and power of OpenCL applications running on a range of distinct heterogeneous platforms as well as to propose a slight extension to the roofline methodology to characterize and track the performance per Watt of an application on a given device. The steps involved in our proposed tool flow can be classified into two stages: (1) to obtain the various device metrics and determine corresponding roofline plots, (2) to measure the metrics of the OpenCL kernel on the device as listed in section 3 and to obtain the derived metrics using the defined formulation in section 3 and combine these with the roofline plot to complete the analysis.

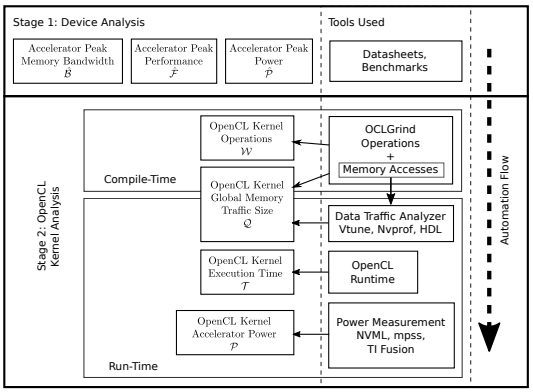

Figure 1: Semi-Automated Tool Flow Design

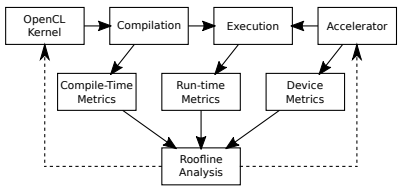

Figure 2: Steps involved in the tool flow

4.1 Stage 1: Device Analysis

As described in fig.1 the first stage is to obtain the device metrics using benchmarks with operations that are relevant to the evaluated OpenCL kernel. Below, we list the basic steps of stage 1 of the proposed methodology:

1. In stage 1, the tool flow starts by measuring peak performance and memory bandwidth of the device using the preset benchmarks providing measurements for $\hat{\mathcal{F}}_f$, $\hat{\mathcal{F}}_i$, $\hat{\mathcal{B}}$ and $\hat{\mathcal{P}}$ for a given device.

2. Assuming a chosen range of operational intensities ($\mathcal{I}_{min}$, $\mathcal{I}_{max}$), the tool flow subsequently determines the performance roofline for this range of operational intensities using eq.7.

3. Similarly stage 1 calculates the performance per Watt roofline using eq.8, which converts peak operations to peak operations per Watt.

$$\min(x \times \hat{\mathcal{B}}, \hat{\mathcal{F}}) \; \forall \; x \in (\mathcal{I}_{min}, \mathcal{I}_{max}) \tag{7}$$

$$\min((x \times \hat{\mathcal{B}})/\hat{\mathcal{P}}), \hat{\mathcal{F}}/\hat{\mathcal{P}}) \; \forall \; x \in (\mathcal{I}_{min}, \mathcal{I}_{max}) \tag{8}$$

4.2 Stage 2: OpenCL Kernel Analysis

The OpenCL kernel analysis represents stage 2 of the tool flow and measures the attainable performance of the OpenCL kernel on a given device. When combined with the device metrics output from stage 1 of the methodology, the output from stage 2 can be used to characterize whether the OpenCL kernel is bound by device memory or device compute performance. Moreover, the outputs of stage 1 and stage 2 can be overlayed on a so-called roofline plot to provide a powerful visual indication as to whether the OpenCL kernel can be optimised further to achieve the optimal attainable performance on the device. Below, we list the basic steps of stage 2 of the proposed methodology:

1. The OpenCL kernel is compiled and the generated intermediate SPIR code is profiled through OCLgrind [17] and the tool calculates ($\mathcal{W}$) for the kernel as well as the memory accesses of the kernel. The memory access information is provided to the data traffic analyzer.

2. In the next step the data traffic analyzer uses platform profile tools to obtain ($\mathcal{Q}$) for the kernel.

3. The tool flow then executes the kernel and calculates its runtime ($\mathcal{T}$).

4. The tool flow relies on high resolution time stamps used during execution and platform specific tools to calculate the kernel's peak power consumption ($\mathcal{P}$).

5. Using eq.1 the tool flow obtains the operational intensity ($\mathcal{I}$) of the kernel, which forms the vertical operational intensity of the kernel on the roofline plot.

6. Using eq.3 and 4 the tool flow obtains the peak performance and performance per Watt of the kernel.

7. Finally, eq.2 is used to calculate the energy consumed by the kernel for a given accelerator.

A complete flow of all the steps involved in the two stages of the proposed tool flow methodology is shown in fig.2, where it can be seen that the metrics are obtained at the compilation and execution phases of the given OpenCL application. In pursuing the experimental measurements as outlined in the stage 1 and stage 2, we are once again inspired by the work in [16]. In particular, we note that measuring the value of $\mathcal{Q}$ can be challenging, even when the relevant hardware counters are available on a given device. In the case that such measurements are simply not feasible (or even warranted), we will work within a range of operational intensities on the horizontal axis bounded by so-called operational intensity "walls", which we can be calculated using static code analysis together with knowledge of the capacity of on-chip memory of the device. Typically, we will assume a "cold" data cache unless stated otherwise and for the case study described later we only consider traffic from and to global memory on the device and not between host memory and device memory. On the latter point, we propose that the methodology as laid out here, can be extended to include data traffic between other data channels, such as network interfaces and PCIe links, which will be included as part of follow up investigations. It should be also be emphasized that we focus our study on OpenCL-based applications exclusively, which implicitly factors in the limitations of the range of OpenCL compilers and runtimes (e.g., respective OpenCL software development environments from Intel, NVIDIA and Xilinx) that we use as part of this project.

5. CASE STUDY

In order to evaluate the proposed tool flow, we have chosen to first focus on various OpenCL implementations of the Bob Jenkins *lookup3* hash function [11], which features in well known datacenter workloads [2]. The pseudo-code for a sequential implementation of the Bob Jenkins function is shown in fig.1. The function processes variable sized keys iteratively in 96bit chunks and each chunk is split into three 32bit numbers, which are added to a set of state variables. Before the next chunk is read, these state variables are mixed using addition, subtraction and XOR operations. Beyond the interest in the *lookup3* function for datacenter applications and its relative simplicity, one reason we have focused on this function is due to the interest in evaluating the proposed method on non-floating point applications, which have not featured heavily as part of roofline analysis to date. The second reason is due to the limited amount of data reuse that the algorithm can avail of, which allows us to make several simplifying assumptions regarding the OpenCL Kernel Global Memory Traffic Size, $\mathcal{Q}$ on each device throughout our analysis.

5.1 Evaluation

In applying the tool flow to OpenCL kernel implementations of the *lookup3* function, we have have been motivated by the heterogeneous systems currently found in the Top500[22] list as well as the growing interest in exploiting FPGAs in hyperscale datacenters [18]. In particular, in this paper, we assess the tool flow on the Intel Xeon Phi 5110P coprocessor, the NVIDIA Tesla K20c and the Xilinx Virtex 7 FPGA-based Alpha Data ADM-PCIE-7V3, which is now supported by the Xilinx SDAccel Development Environment which includes support for OpenCL [24].

Beginning with stage 1 as in fig.1, the tool flow carries out an analysis of each device using adapted versions of the SHOC L0 [6], LINPACK [7] and STREAM[13] benchmarks to obtain values for $\hat{\mathcal{F}}_f$, $\hat{\mathcal{F}}_f$, $\hat{\mathcal{B}}$, $\hat{\mathcal{P}}$ which are reported for each device in table 1. Using these values the roofline diagrams for performance and performance per Watt are presented in figs.3 and 4 respectively.

Device	Theoretical Peak				Measured Peak			
	$\overline{\mathcal{F}_f}$	$\overline{\mathcal{F}_i}$	$\overline{\mathcal{B}}$	$\overline{\mathcal{P}}$	$\hat{\mathcal{F}}_f$	$\hat{\mathcal{F}}_i$	$\hat{\mathcal{B}}$	$\hat{\mathcal{P}}$
	($\times 10^9$OPS/Second)		($\times 10^9$Bytes/Second)	(Watt)	($x10^9$OPS/Second)		($\times 10^9$Bytes/Second)	(Watt)
Tesla K20	3524	587	208	225	2903	585	143	225
Phi 5110P	1988	1988	320	245	1189	946	119	245
ADM 7V3	738	8880	21	25	200§	3032‡	8.5†	25

Table 1: Device analysis summary of the platforms investigate in this report.

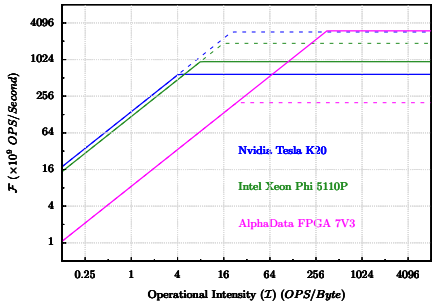

Figure 3: Performance roofline comparison for all devices. Dashed lines represent floating point operations. Solid lines represent integer operations.

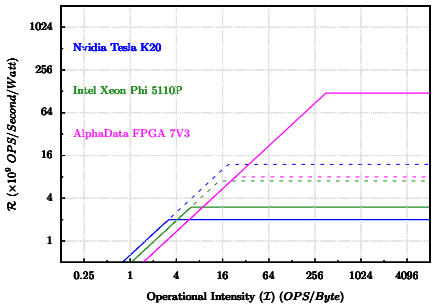

Figure 4: Performance per Watt comparison for all devices. Dashed lines represent floating point operations. Solid lines represent integer operations.

ALGORITHM 1: Bob Jenkins `lookup3` hash function

```
key ← Input string to hash
length ← Length of input string
init ← Initialization value of the hash
hash → Returns the hash value
begin
    a, b, c ← Initialize based on length and init
    index ← Index of the key
    while length > 12 do                                    // mixing
        a += key[ index + 0 ]
        b += key[ index + 1 ]
        c += key[ index + 2 ]
        Mix ( a, b, c )
        length −= 12
        index −= 3
    end
    /* Mix the remainder in a, b, c                                */
    /* and return the hash                                         */
    return MixRemainder ( a, b, c, key, length )
end
```

Following stage 1, the next stage of the tool flow involves the analysis of an unoptimised "out-of-the box" OpenCL implementation of the *lookup3* function (closely aligning with the pseudo-code in alg.1), in this case making use of OCLGrind, an open source SPIR interpreter and device simulator, which can be used to count the SPIR instructions of an OpenCL kernel. The output of OCLGrind when applied to the out-of-the-box OpenCL implementation of the *lookup3* function is shown in fig.5. By counting all of the relevant operations (in this case "`add,xor,sub,shl,lshr,or, getelementptr,icmp,mul,and,div`") selected by the user the value of $\mathcal{W}$ can be easily determined and in this case we find

```
Instructions executed for kernel 'hash':
     242,802,730 - add
     175,687,220 - phi
     166,657,332 - xor
     158,268,724 - sub
     158,268,724 - shl
     158,268,724 - lshr
     158,268,724 - or
     100,506,735 - getelementptr
      83,568,763 - load global (334,275,052 bytes)
      66,951,596 - br
      33,556,176 - icmp
      25,165,824 - mul
      16,777,216 - zext
      14,559,207 - and
       8,388,608 - udiv
       8,388,608 - trunc
       8,388,608 - switch
       8,388,608 - select
       8,388,608 - ret
       8,388,608 - store global (33,554,432 bytes)
       8,388,608 - call get_global_id()
```

Figure 5: OCLGrind output on the *lookup3* kernel with 8 million random keys of size 0 – 60 bytes

that $\mathcal{W} = 1224$ Million OPS. As mentioned previously, the *lookup3* function avails of no data-reuse, so that there is a one-time fetch-and-process of each key from global memory to on-chip memory, representing a best-case scenario of one compulsory miss per key, which considerably simplifies our analysis and which strongly suggests that we can also make use of OCLGrind to obtain a value for the $\mathcal{Q}$, here. In this case OCLGrind calculates a value of $\mathcal{Q} = 367$ Million bytes which aligns exactly with our own static code analysis carried out by hand. With these values, the tool flow calculates a value of $\mathcal{I} = 3.33$ $OPS/byte$ based on eq.1 for the out-of-the-box OpenCL implementation of the *lookup3* function, which provides enough information to overlay an operational intensity wall on the roofline plot for each device, representing an upper bound on attainable performance on each device for this particular kernel and also indicating whether the kernel is compute or memory bound on a given device.

As part of subsequent repeatable steps, optimized implementations of the OpenCL *lookup3* kernel for each device is analyzed using the tool flow where values of $\mathcal{T}$ and $\mathcal{P}$ are obtained for each optimized implementation on each device and are plotted as horizontal lines on the roofline plots, clearly indicating the difference between achieved and attainable performance on the given device. In sections 5.1.1, 5.1.2, 5.1.3, we provide a brief overview of of how we have applied the full tool flow to analyze target-specific optimizations of the *lookup3* kernel on each of our three devices. It should be emphasized at this point, that the focus of this paper is purely on how the overall tool flow might be applied to analyze OpenCL applications across a diverse range of heterogeneous devices and is not focused on achieving optimal performance on a given device.

† ADM 7V3 bandwidth is estimated assuming 40% access efficiency for random access to DDR3 DRAM

§ ADM 7V3 peak floating point performance is measured using an inhouse FPGA microbenchmark

‡ ADM 7V3 peak integer performance is estimated using, 70% of (#LUTS/20) *200Mhz(operating frequency of the FPGA), which is 0.7*(433200/20)*200 = 3032.4 OPS/s. Remaining LUTs comprise infrastructure surrounding kernel.

5.1.1 *Intel Xeon Phi 5110P*

The first device we apply the tool flow to is the Intel Xeon Phi 5110P coprocessor, which is a symmetric multiprocessor in the form factor of a PCI express device. The 5110P coprocessor features 60 cores clocked at 1.053 GHz, supporting 64-bit x86 instructions. Due to the low-clock frequency of each core, in order to achieve best performance, the developer needs to make effective use of the 512bit wide vector processing units per core on the device. In figs.6 and 7, we see the first of our roofline plots for performance and performance per Watt respectively, as produced by our proposed tool flow, which shows the roofline of the Xeon Phi 5110P for integer-based operations, as well as the overlayed vertical operational intensity wall for the out-of-the-box OpenCL implementation of the *lookup3* function. Fig.6 strongly indicates that an optimal implementation of the function is memory bound on the Xeon Phi. Subsequent to parallelizing the function over OpenCL work groups, which in this case map closely to available hardware threads on the Xeon Phi, as well as carrying out optimizations on array access patterns to reduce indirect referencing, we achieve a performance of 66.70×10^9 OPS/second, shown by the red line in fig.6 and a performance per Watt of 0.38×10^9 OPS/second/Watt, shown in 7. Already, at this stage, fig.6 clearly indicates that there is a substantial gap between the performance we achieve with our optimized implementation and the attainable performance suggested by the operational intensity wall. While the roofline approach is somewhat inadequate in determining the reasons for this performance gap, there are indications that performance could be improved by more effective use of the vector processing units on the Xeon Phi, which we make limited use of due to the inherent feedback loop and branch divergence found in the *lookup3* function. However, we admit that this does require further analysis, which is currently beyond the scope of this paper.

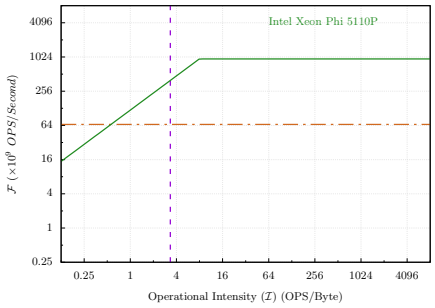

Figure 6: Performance Roofline – Intel Xeon Phi 5110P

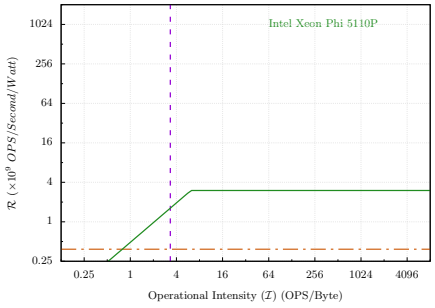

Figure 7: Performance Per Watt Roofline – Intel Xeon Phi 5110P

5.1.2 *Nvidia Tesla K20*

The second device we apply the tool flow to is the NVIDIA Tesla K20c, which is a general purpose GPU in the form factor of a PCI express device and is based on the Kepler GK110 core. There are 192 cores in each Streaming Multiprocessor (SMX) processing engine. The K20 actually contains 15 SMX engines, although only 13 are available. This gives a total of 2,496 available cores, with two operations per clock cycle, running at a base frequency of 706 MHz. In regard to integer operations, each SMX of the Kepler architecture supports only 32 operations per clock cycle for 32-bit multiply and multiply-and-add operation. Once again, the overlayed vertical operational intensity wall for the out-of-the-box OpenCL implementation of the *lookup3* function indicates that the an optimal implementation of the function is memory bound on the Tesla K20. Again, subsequent to parallelising the function over OpenCL work groups, which in this case map closely to the SMXs on the device, as well as carrying out optimizations to improve memory coalescing, we achieve a performance of 126.42×10^9 OPS/second, shown by the horizontal line in fig.8 and a performance per Watt of 1.18×10^9 OPS/second/Watt, indicated by the horizontal line in fig.9. Again, a suggested explanation for the performance gap on this device could possibly be the branch divergence found in the *lookup3* function, which will impede performance on the GPU, but this requires deeper analysis.

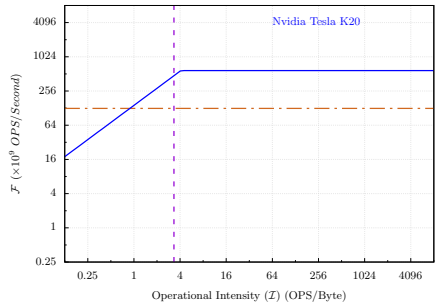

Figure 8: Performance Roofline – Nvidia Tesla K20

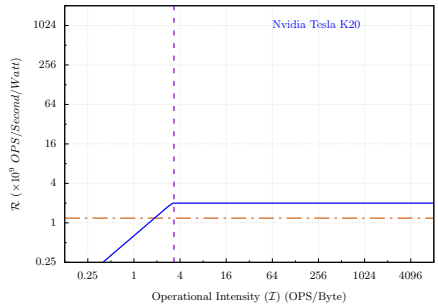

Figure 9: Performance Per Watt Roofline – Nvidia Tesla K20

5.1.3 *Alpha Data ADM-PCIE-7V3*

The final device we apply the tool flow to is the the Alpha Data ADM-PCIE-7V3, a high performance reconfigurable half-length, low profile x8 PCIe form factor board based on the Xilinx Virtex-7 V7690T FPGA . For our analysis of theoretical peak throughput, we assume that 1 SP floating point instruction takes 2 DSPs, 150 LUTs and clocks at 410 MHz. The V7690T has 3600 DSPs and 433,200 LUTs thereby offering theoretical peak floating throughput of 738 GFLOPS/s. For 32bit integer operations, we assume that 1 operation occupies roughly 20 LUTs and clocks at 410 MHz which is

extrapolated from in house benchmarks*, offering a theoretical peak integer throughput of 8880 GOPS/s. For integer throughput, we assume that the available region of the device for acceleration is 70% of the overall LUT resources, and then apply the 20 LUT per integer operation assumption as above. We averaged in our experiments 20 LUTs per operation, in future work we intend to expand this by generating application specific rooflines that provide a more accurate estimate. In regards to frequency, the current version of SDAccel clocks the accelerator region at 200 MHz, resulting in a more realistic value of attainable peak integer throughput of 3032 GOPS/s. In this case the overlayed vertical operational intensity wall for the out-of-the-box OpenCL implementation of the *lookup3* function indicates that an optimal implementation of the function is memory bound. On the FPGA, we performed optimizations that included pipelining and double buffering to significantly improve performance relative to the out-of-the-box implementation. We achieve a performance of 18.11×10^9 OPS/second, shown by the horizontal line in fig.10 and a performance per Watt of 1.02×10^9 OPS/second/Watt, indicated by the horizontal line in fig.11. Fig.10 shows that we are achieving close to attainable performance on this device, indicating that further performance improvement can only come as a result of improvements in available memory bandwidth of the device.

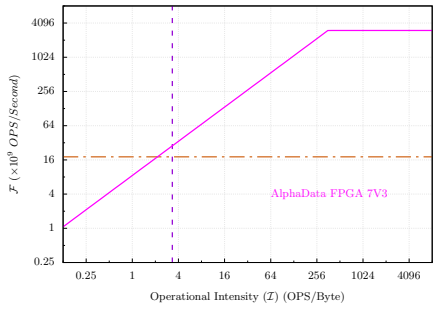

Figure 10: Performance Roofline – AlphaData 7V3

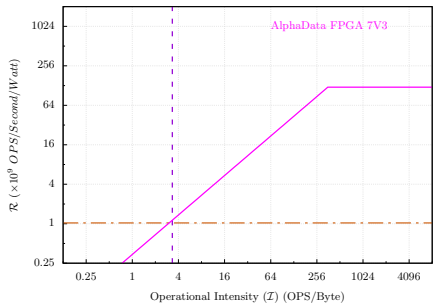

Figure 11: Performance Per Watt Roofline – AlphaData 7V3

5.1.4 *Energy Consumption Comparison*

Since performance and performance per Watt can be expressed as a factor of operational intensity, we were able to represent them using roofline models. However, energy consumption of a device is a crucial metric of a kernel. To represent this we present a graph shown in fig.12 that gives us the absolute energy consumption of the device for the evaluated optimized implementations as described in sections 5.1.1,5.1.2 and 5.1.3 from using eq.2 and $\mathcal{T}$ and $\mathcal{P}$ for each device. In the future we would like to extend the work by Choi et al. [4] to apply it devices such as the FPGAs as well.

*Contact Xilinx for more information

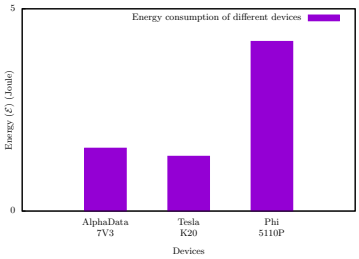

Figure 12: Energy

6. RELATED WORK

Choi et al. [4] describe an energy based roofline model that provides insight into relationship between time, energy and power for an algorithm. They show the balance gap between the traditional time-balance and their energy-balance and how it can be used to determine peak energy efficiency of the algorithm. Our methodology proposes a tool flow design that can semi-automatically measure and compare different devices using roofline models. We also propose performance per Watt extension to roofline model for comparing power consumption of different devices. In the future we would like to include Choi et al.'s energy roofline into our tool flow design to provide a more comprehensive anaylsis.

The Spiral Project[16] provides a methodology on how experimental values can be measured and applied on otherwise theoretical roofline models. In our tool flow design we leverage the roofline model to compare a range of diverse hardware architectures. Also, we provide techniques on how integer computations and various other measurments can be obtained on a range of OpenCL accelerators and compared using roofline model.

Long established benchmark suites such as NASA NAS [1], LINPACK [7] and STREAM [13] provide performance data for systems composed of traditional CPU. With the advent of accelerated computing, several packages have emerged to evaluate heterogeneous platforms. Amongst them are Rodinia[3], SHOC[6] and OpenDwarfs[9]. These suites provide several kernels representing various domains within scientific computing. Kernels are implemented using standard parallel frameworks such as OpenMP, CUDA and OpenCL. The targets architectures for which these suites are optimized are multicore and GPU based systems, therefore they are not suitable for comparing a broader range of devices. While we use some of these benchmarks in stage 1 of our tool flow to obtain device metrics, our methodology is targeted towards comparison of performance and energy efficiency of an OpenCL application across various devices rather than benchmarking the accelerator itself.

7. CONCLUSION

In our view, the most interesting outcome of our work is the viability of a semi-automated tool that can benchmark, measure and evaluate implementations of an algorithm across different OpenCL accelerators. Our proposed performance per Watt extension to roofline models presents insight into the peak energy efficiency of the device for a given OpenCL kernel.

We are currently investigating a diverse range of OpenCL applications that reflect a wide range of operational intensities. By also including figures of merit on productivity and code complexity, we hope to shed more light on the advantages and disadvantages of programming models, tools and hardware platforms in the near future. An example of such insight already comes from an early stage analysis of an out-of-the-box OpenCL implementation of the bioinformatics Smith-Waterman algorithm[20] with our tool flow, we see that such an integer-based kernel with a high operational intensity ($\mathcal{I}$=75) would perform best on the FPGA-based Alpha Data platform investigated in this paper as is clearly evident from the roofline plot comparison of devices in fig.3.

In providing experimental results on otherwise theoretical roofline models, there are several challenges. In this paper we have proposed some techniques involved in obtaining "OpenCL Kernel Operations" and "OpenCL Kernel Global Memory Traffic Size" for obtaining operational intensity. However, obtaining accurate energy measurements proved to be a challenge. As future work we are developing a fine-grained measurement facility at ICHEC that can accurately profile energy consumption of multiple accelerators with a high temporal resolution to address this issue.

In spite of these limitations, we hope algorithm designers, performance engineers, and architects will find the proposed tool flow an interesting starting point for discussion on providing a means of understanding bottlenecks and tracking attainable performance across a wider range of OpenCL supported heterogeneous platforms.

References

[1] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, et al. The nas parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[2] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4): 485–509, 2004.

[3] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54. IEEE, 2009.

[4] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc. A roofline model of energy. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 661–672. IEEE, 2013.

[5] T. S. Czajkowski, U. Aydonat, D. Denisenko, J. Freeman, M. Kinsner, D. Neto, J. Wong, P. Yiannacouras, and D. P. Singh. From opencl to high-performance hardware on fpgas. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 531–534. IEEE, 2012.

[6] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The scalable heterogeneous computing (shoc) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU-3, pages 63–74, New York, NY, USA, 2010. ACM.

[7] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK users' guide*, volume 8. Siam, 1979.

[8] W. Feng and K. W. Cameron. The green500 list: Encouraging sustainable supercomputing. *Computer*, 40 (12):50–55, 2007.

[9] W. Feng, H. Lin, T. Scogland, and J. Zhang. Opencl and the 13 dwarfs: a work in progress. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 291–294. ACM, 2012.

[10] Intel Corporation. Intel manycore platform software stack. https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss, September 2015.

[11] B. Jenkins. Bob jenkins lookup3. http://burtleburtle.net/bob/c/lookup3.c, May 2006.

[12] Khronos Group. Opencl standard. https://www.khronos.org/opencl/, September 2015.

[13] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, Dec. 1995.

[14] NVIDIA corporation. Nvidia visual profiler, 2011.

[15] NVIDIA corporation. Nvidia management library. https://developer.nvidia.com/nvidia-management-library-nvml, September 2015.

[16] G. Ofenbeck, R. Steinmann, V. C. Cabezas, D. G. Spampinato, and M. Püschel. Applying the roofline model. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 76 – 85, 2014.

[17] J. Price and S. McIntosh-Smith. Oclgrind. https://github.com/jrprice/Oclgrind, September 2015.

[18] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Xiao, and D. Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 13–24, June 2014. doi: 10.1109/ISCA.2014.6853195.

[19] J. Reinders. *VTune performance analyzer essentials*. Intel Press, 2005.

[20] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[21] Texas Instruments. Fusion digital power designer. http://www.ti.com/tool/fusion_digital_power_designer, September 2015.

[22] Top500.org. Top500 TIANHE-2 or MILKYWAY-2 Fastest Supercomputer. http://www.top500.org/system/177999, Jun 2015.

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

[24] Xilinx. Sdaccel development environment. http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html, 2015.