Performance Evalulation of a Heterogenous System and CPU Platform Using a Data Transform Application

Clayton Faber, cfaber@wustl.edu (A paper written under the guidance of Prof. Raj Jain) (A paper written under the guidance of Prof. Raj Jain)

Download 

Abstract:

Heterogeneous computing has grown popular as of late as a way to scale up computing power without cranking up the clock speed and FPGAs have become quite popular in the heterogeneous field as it is a low power flexible computing device. Taking advantage of this the new Intel/Altera group have created a new type of heterogeneous system that implements a Xeon core chip with a closely interconnected, cache coherent bus called HARP. This setup make an excellent target for data transformation problems that could be preformed by the FPGA and potential have a faster execution time than a standard CPU build. In this paper a data transformation app is implemented in a sequential CPU environment, a parallel CPU OpenMP environment, and two OpenCL environments to use the FPGA as the main computation device. These runs are compared with each other to see if there any benefit between the execution models. Tests are ran with large file sizes as input data into the transformation from 512KB to 512MB to hopefully paint a clear picture of how execution works on the HARP.

Keywords: Keywords: Heterogeneous Computing, OpenCL, HLS, Intel HARP, FPGA, OpenMP, Data Transform, Stream Processing

Table of Contents:

1. Introduction

In an effort to solve current computer science problems, heterogeneous computing has been growing as an answer in the face of Dennard Scaling and Moore's law slowing the progress of computer architecture improvements [Zahran'16]. Heterogeneous computing covers many different facets of the computer architecture landscape such as multicore processors, graphical processing units, and other specific processors and attempts to marry them into a system that can hopefully leverage the best aspects about each unit for maximum throughput and computing power. One such facet of heterogeneous computing that has become exceedingly popular in the recent years are field programmable gate arrays (FPGA(s)).As the name implies they are re-configurable chips that can implement hardware based on some specification provided by the programmer. Although the chip excels in versatility and power efficiency, it is often hard to program for and suffers from long memory transaction times [Hussain'14].

Recently, with the merger of Altera, one of the world's largest FPGA manufactures, and Intel, the purveyor of the x86 architecture, there has been an attempt to remedy these issues with Hardware Accelerator Research Program (HARP). The system proposed in this program combines an Intel processor and a closely interconnected Altera FPGA so that the two can work in tandem for processing tasks. One such task that this system is being targeted for is stream processing of data transformations.

In most cases data must first be preprocessed before it can go off to its final "destination" where it is potentially presented to an algorithm or stored in a database. The HARP system makes a tantalizing target for stream processing as the transformation itself could be implemented in hardware with the added bonus of close interconnectivity between the Central Processing Unit (CPU) and FPGA. However, traditional methods of creating FPGA configurations is usually done in a hardware description language (HDL) which can be difficult and time consuming to design and debug. To this end, Altera has created a high level synthesis (HLS) tool using the OpenCL language to generate hardware implementations of kernels[Intel/Altera'17].

In order to make more concrete statements about the cost of using OpenCL for HLS and to examine the performance of the HARP platform in a stream processing scenario, this paper sets out to evaluate a data transform application that reads data, performs a data transformation (16-bit fixed point number to 32-bit floating point number), and then stores it for later use. Although the program and transformation will be simple, the goal is to measure execution time of the data transform and the total system in the face of large amounts of data. As a foil to this experiment a CPU version of the program will be measured that can execute in a sequential or multi-threaded fashion with the help of OpenMP C libraries. The goal of this work is to convince a reader that the potential for FPGA acceleration is realizable now more than ever with the advent of new tools and impressive hardware found in the HARP platform. Section 2 begins with an

introduction to the HARP system and OpenCL HLS tools, followed by a discussion on other work in the field. Section 3 will detail the experiment breaking down the program and platforms used as well as the factors and techniques used for data analysis. Section 4 discusses the results and what information can be learned. Finally, concluding with an overall summary of the work will appear in section 5.

2. Background and Other Work

The goal of this project is to try and implement a data transform that would be used in preprocessing step, potentially before being used in its final application. This section serves as an introduction to concepts and tools that will be used in this application along with some background and work related to this experiment.

2.1 Stream Processing and Heterogeneous Computing

In a stream processing application desired computations are split into kernels which feed into each other [Beard '17]. A data transformation can be a portion of the stream processing application where data is ingested, crunched and sent off to the next stage (i.e. an algorithm or record). When running data transforms most operations end up being simple, usually taking the form of data truncations, bit shifting, data replacement, and others. Often these programs can take multiple CPU days due to their tedious nature of constantly reading in new data, transforming it, then shipping it away. The action itself may not take much time or have an incredibly complex implementation but it burns time in read and write cycles which are required for every transformation.

These transformations make excellent candidates for offloading to specialized hardware on a heterogeneous system to hopefully perform them faster than a standard CPU could. In heterogeneous systems a main compute unit and have one or more specialized units such as a graphical processing unit (GPU) or a coprocessor that can be used to offload computation. In the HARP system's case the specialized compute unit is an FPGA which can be used to implement almost any type of hardware and use it for co-processing.

A basic block diagram of the HARP system is available in figure 1 and shows the connectivity between the CPU (Intel Xeon), memory, and FPGA (Arria 10 GTX1150). In the HARP system the FPGA and CPU are connected via a cache coherent interface which facilitates quick memory transactions between the two. Offloading computation to an FPGA also frees up the main core to perform other tasks while potentially waiting for proper data to perform an action on. The process of programming an FPGA for a specific task using traditional methods can be a time consuming endeavor, as typically it is written in a virtual hardware description language (VHDL) like Verilog.

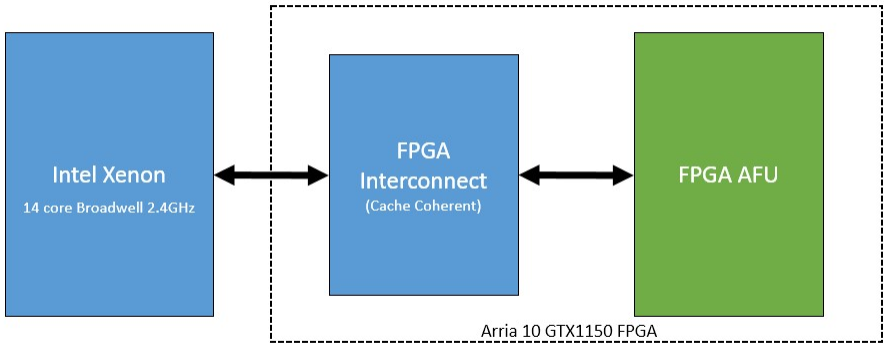

Figure 1: A block diagram of the HARP system

Although Verilog and other VHDL languages are quite powerful programming tools for FPGAs it is usually their complexity that becomes the barrier for FPGA implementation. In a VHDL world a programmer has to be concerned with what happens on every clock pulse and stitching up or authoring modules to implement hardware. While this is typically managed in some sense it can quickly become an overwhelming task and could even run slower than a CPU if implemented poorly. In an attempt to combat these issues a number of high level synthesis (HLS) tools are in development in order to make the process of programming a slightly less painful endeavor for the average programmer and requiring less knowledge about the intricate details of FPGA systems.

2.2 OpenCL High Level Synthesis

An HLS tool used to target the HARP system is the "Intel FPGA SDK for OpenCL" which allows for the compilation of OpenCL kernels into VHDL which is then used to generate the programming file for the FPGA itself. The OpenCL language is a framework designed specifically for heterogeneous systems to target GPUs and CPUs while remaining agnostic to the target Programs are split into a host program which is in charge of loading the framework and a kernel which contains the code meant to do the heavy lifting of the program. As long as the hardware is OpenCL compliant the kernel will execute on whatever target the programmer desires, a CPU, a GPU, or an FPGA in this case. Being a subset of ISO C99, the hope is that programmers can quickly pick up the language and start deploying kernels for their given application [Khronos '17].

In figure 2 the breakdown of the process compiling an OpenCL program for the HARP is shown. Here a user creates both a Main.cpp which contains the host program and a UsrKernel.cl which contains the kernel to be offloaded to the FPGA. The UsrKernel.cl is compiled using the Altera OpenCL Compiler (aoc) compiler which creates multiple files of VHDL describing how the kernel is implemented in hardware. The compiler then calls Quartus, Altera's VHDL compiler, to create the FPGA programming file. The basic Main.cpp program is compiled with the standard

g++ compiler linked with Altera's FPGA OpenCL libraries. When running the host program a call is made to load in the compiled binary to program the FPGA which is denoted by the dotted line. The main program handles the setup of the context for the kernel and queues it for execution.

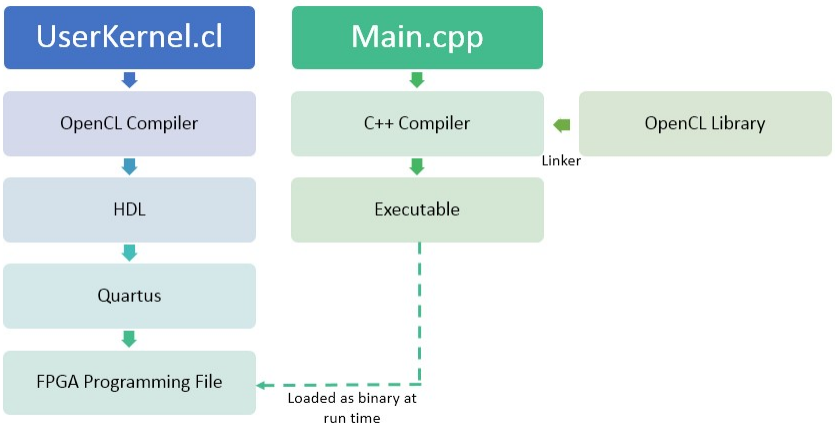

Figure 2: OpenCL programming diagram

Using the HARP in tandem with the OpenCL HLS tools a programmer is able to author code and quickly start working with the FPGA directly without worrying about things like low level memory management and creating the system to handle memory transactions. In a stream processing application a programmer can quickly reap the benefits of hardware designed specifically to crunch through data transformations and not worry about the complexities of authoring VHDL for their simple task. One thing that is important to note however is that this method of programming requires the author to spend time running a separate compilation for the VHDL output that is offline compared to the encouraged run-time compilation of normal OpenCL applications [Stone '10].

2.3 Related Work

Of course, HARP is not unique in using FPGAs to achieve performance benefits. In a large scale datacenter setting FPGAs can be shown to have a 95% overall performance gain over a software approach when it comes to a web page ranking task for a search platform. [Putnam '14] With heterogeneous computing systems an FPGA implementation of a Bayesian network was shown to have a significant performance gain (x4.18) over general-purpose GPUs implementing the same problem. [Fletcher '10] However, in both these cases the implementations were programmed using fine-grained VHDL programming languages, which may not speak to general performance gain using HLS tools. When using OpenCL as an HLS tool one group found they could not quite match the speed of a GPU in their tests but had a better power efficiency overall [Zohouri '16] but it should be noted that the code was originally designed to run on GPUs and not FPGAs. This group also found that in most cases the FPGA did in fact beat out the CPU in terms

of performance in three of its five cases but these were evaluating the OpenCL kernel running on a CPU and not an OpenMP implementation used here in this experiment.

Work has been done in an attempt to evaluate OpenCL as a HLS tool through different models of performance. One group produced a benchmark called CHO (CHStone OpenCL implementation) but had some mixed results resulting in some of their kernels not synthesizing properly or producing incorrect results [Ndu '15]. The group also acknowledges that the straight porting of GPU code to FPGA, even when the compilation works, is not a guaranteed way to see performance gains and indeed they found that half of the kernels ran faster and half ran slower. Y. Luo et al. evaluated the existing XSBench in OpenCL to try and evaluate FPGA implementations of kernels that have irregular memory access patterns [Lou '17]. Their findings pointed to a general 35% performance loss when using the FPGA over a CPU. While this may start sounding like an FPGA is outclassed when it comes to OpenCL applications most examples using OpenCL were ported directly from existing GPU code and may not translate well to the FPGA space. Obviously, some comparisons between this experiment and the related work will not hold as the HARP system is radically different than any experimental setup seen here because of its cache coherent interface.

3. Experiment Design

Diving into the details of the experiment, the factors are described and a general breakdown of how the program is setup is presented. The major differences between the OpenMP and OpenCL implementation are described here and a run down of the specs of the system specifications are presented.

3.1 Experimental Details

For the experiment a simple data transform application will be measured, that is, an application that solely transforms data into a proper format. The application under evaluation, Fix_To_Float, is meant to mimic a transaction of raw fixed point data which needs to be transformed into a floating point value so that some more complex math can be performed later down the road. This program evaluates fixed point number that are 16 bits in size and converts them to a 32 bit float representation essentially doubling the total size it will occupy in memory. This application has a sequential access pattern which should translate into a parallel implementation quite nicely resulting in a faster run time.

Execution time is one of the most important metrics in system evaluation and in these experiments a lower execution time will be the indicator of merit. In a setup like this there are quite a few parameters to play around with such as execution type, file size, kernel scheduling, hardware, and compiler optimizations, for these measurements the only factors that will be measured are the input file size and the execution type. The execution type will have four levels: 1. A sequential CPU version that evaluates each transformation one at a time. 2. A OpenMP version that spawns as many threads as there are physical cores (14 in this case) and uses them to compute a parallel for loop. 3. A naive OpenCL FPGA design that implements the kernel

without any sort of vectorization or parallel specification. 4. A Single-Instruction Multiple Data (SIMD) OpenCL FPGA implementation that specifies a SIMD width of 16 items with a work group size of 64. This specification allows the FPGA to execute on 16 items at a time within a work group resulting in work being spread across 4 hardware threads. The data size will have eleven levels of file sizes ranging from 512KB to 512MB by increasing powers of two (512KB, 1MB, 2MB, 4MB, etc.). Table 1 has a comparison for the data size and the number of data points that are transformed. For completeness sake, each experiment will be replicated three times and then averaged. The system specs of the HARP system as reported by **/proc/cpuinfo** and **free** are shown in table 2 the OpenMP version of the program will not use the FPGA while the OpenCL version will utilize both the CPU and FPGA.

Input Size	# of Bits	# of Fixed Pt. Values
512KB	4.10E+06	2.56E+05
1MB	8.19E+06	5.12E+05
2MB	1.64E+07	1.02E+06
4MB	3.28E+07	2.05E+06
8MB	6.55E+07	4.10E+06
16MB	1.31E+08	8.19E+06
32MB	2.62E+08	1.64E+07
64MB	5.24E+08	3.28E+07
128MB	1.05E+09	6.55E+07
256MB	2.10E+09	1.31E+08
512MB	4.19E+09	2.62E+08

Table 1: The relationship between the input file size and the number of fixed point numbers

Intel HARP CPU and Memory info	
Clock speed	2.40 GHz
Number of cores	14 Physical, 28 Logical
Cache Size	35 MB
Main Memory Size	64 GB

Figure 2: CPU info of the HARP

This experimental design follows a two-factor full factorial design with replications and can be used to determine the effect of the two factors chosen in this experiment [Jain '91]. While it should be trivial to show that the data set size is a major factor in this experiment as the computation depends on the number of elements it will be interesting to observe how much of an effect the different execution models have on the running time.

3.2 Program Details

The execution path for the program is a follows: 1. Any necessary setup and/or bookkeeping is performed 2. A binary file is read into a memory allocated buffer for the program. 3. A similar sized buffer is then created for the float values to be saved by the computation. 4. The buffers are then used in the transformation, a combination of type-casting, bit shifting and division performs the conversion from a fixed point number to a floating point number. 5. Each value is saved into the float buffer as it is computed 6. After all transforms are complete the program is complete and stops the timers. 7. Verification is performed to make sure the fixed and float values agree. Our main measure for this program is execution time of which there are two specific time regions that are looked at: 1. the time from 1-6 (system execution) and 2. The execution time of 4-6 (data transform). Although the verification step is necessary, it is not a time slice of the

program that is interesting as it has nothing to do with memory movement or data transformation and is outside the scope of this experiment.

Listings 1 and 2 present a pseudocode representation of the program. In the OpenCL representation shared virtual memory (SVM) buffers are used to pass data to and from the FPGA kernel. SVM memory buffers are new to OpenCL 2.0 and allow the host program and FPGA kernel to share pointers [Intel/Altera '14,]. This eliminates the need to copy data from a host buffer into a device buffer creating a much easier interface to work with on the host side. There's a large amount of overhead in the OpenCL version of the program because of the necessary buffer initialization, context setup, and error checking required by OpenCL. In the OpenMP implementation threads are spawned using a **#pragma omp for** call putting the limit of threads at 14 which matches the number of physical cores on the HARP machine. The for loop is the only portion of the code where parallel execution is used and the rest is sequential. Of course, the sequential version of the program has no OpenMP pragma and executes sequentially the entire time. Listing 3 shows the actual data transform as it is implemented in both versions of the code. To access the array the OpenCL version uses a **get_global_id()** call to return the specific index of the work item and in the OpenMP version the for loop counter is used. Finally in listing 4 the extra attributes that are added to the OpenCL kernel are shown which declare a 3-dimensional workgroup size of 64x1x1 (creating a total workgroup size of 64) and a SIMD width of 16 to split the workgroup onto 4 different hardware threads on the FPGA.

```
main_OpenCL(){
  start system execution time
  create OpenCL context for Harp
  create an SVM memory buffer for fixed point numbers
  read binary file into SVM memory buffer
  create a float buffer of similar size
  create a command queue for kernel
  load in FPGA programming file and build kernel
  set kernel arguments
  start data transformation timer
  start kernel execution
  while(kernel != done){
      wait();
  }
  stop data transformation timer
  un-map buffers
  stop system execution time
  verify
  return 0
}
```

Listing 1: OpenCL pseudocode

```
main_OpenMP(){
   start system execution timer
   create fixed point buffer using malloc
   create float buffer using malloc
   start data transformation timer
   BEGIN: parallel section
       for each value in fixed point buffer
           floatbuf[i] = fixtofloat(fixed[i], Q)
   END: parallel section
   Stop data transformation timer
   Stop system execution timer
   verify
   return 0
}
```

Listing 2: OpenMP pseudocode

```
float_buf[gid] = ( ((float)(fix_buf[gid])) / ((float)(1 << Qval )) );
```

Listing 3: The data transformation as implemented in code

```
__attribute__((num_simd_work_items(16)))
__attribute__((reqd_work_group_size(64,1,1)))
__kernel void ...
```

Listing 4: Extra attributes added to the OpenCL kernel

4. Experimental Results

As mentioned in the previous section this experiment is broken into two measures: the running time of the system and the running time of the data transform. This section presents the results of both measurements and compares them using analysis of variance (ANOVA). The final two graphs in this section plot the system and data transform running times as a function of the data set size. This section will conclude with a summary discussing some of the results of the experiment. It is important to note that all samples, except for the final two graphs at the end, have gone through a log transformation due to the large ratio between the highest and lowest numbers in the data set.

4.1. Data Transform

Log of Data Transform Time in Milliseconds (Averaged from Three Replications)

	CPU Seq.	CPU OMP	FPGA Naïve	FPGA SIMD	Row Sum	Row Avg.	Row Effect
500KB	0.34095	-0.12127	0.44476	0.14828	0.81273	0.20318	-1.13587
1MB	0.63679	0.10811	0.59026	0.25966	1.59482	0.39871	-0.94034
2MB	0.88970	0.44797	0.78589	0.31782	2.44137	0.61034	-0.72871
4MB	1.18253	0.72570	1.01600	0.35188	3.27609	0.81902	-0.52003
8MB	1.50578	0.94042	1.27995	0.49441	4.22055	1.05514	-0.28391
16MB	1.73842	1.18611	1.56258	0.61445	5.10155	1.27539	-0.06366
32MB	1.94743	1.45113	1.85233	0.82648	6.07738	1.51935	0.18030
64MB	2.19173	1.76170	2.14771	1.06453	7.16567	1.79142	0.45237
128MB	2.49151	2.02279	2.44586	1.33484	8.29501	2.07375	0.73470
256MB	2.79222	2.26725	2.74548	1.61506	9.42001	2.35500	1.01595
512MB	3.09292	2.46540	3.04586	1.90879	10.51297	2.62824	1.28919
Col. Sum	18.80999	13.25531	17.91667	8.93620	58.91816		
Col. Avg.	1.71000	1.20503	1.62879	0.81238		1.33905	
Col. Effect	0.37095	-0.13402	0.28974	-0.52667			

Interactions Between Execution Platform and Data Size

	CPU Seq.	CPU OMP	FPGA Naïve	FPGA SIMD
500KB	-0.23318	-0.19043	-0.04816	0.47177
1MB	-0.13287	-0.15657	-0.09818	0.38762
2MB	-0.09159	-0.02836	-0.11420	0.23414
4MB	-0.00745	0.04069	-0.09276	0.05952
8MB	0.07969	0.01930	-0.06493	-0.03406
16MB	0.09208	0.04474	-0.00255	-0.13427
32MB	0.05714	0.06581	0.04325	-0.16619
64MB	0.02937	0.10430	0.06656	-0.20022
128MB	0.04681	0.08306	0.08237	-0.21224
256MB	0.06627	0.04627	0.10074	-0.21328
512MB	0.09373	-0.02882	0.12788	-0.19278

Table 3: The effects and interactions table for the Data Transform

In table 3 the computation of effects for the data transform are shown. The average running time for a data transformation in this spread results in a 1.34 log millisecond execution time (21 .88 ms). Each file size results roughly in a ~.24 log millisecond spread between each other (~1.5 milliseconds). The execution model has fairly large gaps between the implementations where in cases of CPU vs FPGA a gap is as wide .8 log milliseconds (6.76 ms) in the worst case scenario. Most interactions in table 3 result point to SIMD FPGA having the better on average performance between either the file size or the execution mode as a majority of its interactions are less than zero.

ANOVA Table						
Componet	SSQ	Percentage of Variation	Degrees of freedom	Mean Square	F-Compute	F-Table
y…	334.98074		132			
ȳ...	236.68294		1			
y...-ȳ...	98.29779	100.00%	131			
A	17.05745	17.35%	3	5.68582	6715.51229	2.72
B	78.37978	79.74%	10	7.83798	9257.42778	1.94
AB	2.78605	2.83%	30	0.09287	109.68672	1.59
error	0.07451	0.08%	88	0.00085		

Table 4: ANOVA table for data transform

Moving onto the ANOVA table for the data transformations (figure 4) we see that effect B, the data set size, has the largest effect on the variation between the runs, which may prove the hypothesis correct that the data set size is the major factor in these transformations. The next factor that explains 17.35% of the variation is the execution mode followed by the interactions at 2.83%, although this is less than a quarter of the total variation it still could be significant. The variation due to errors is very low thankfully, with a .08% variation.

Confidence intervals for the Effects of Data Size and Platform at 99% confidence

Parameter	Mean Effect	Std Deviation	Confidence Interval
μ	1.33905	0.02910	
Processing Platform			z= 2.576
CPU Sequential Exec.	0.37095	0.00439	**(0.35965, 0.38225)**
CPU OpenMP Exec.	-0.13402	0.00439	**(-0.14532, -0.12272)**
FPGA Naïve Ver.	0.28974	0.00439	**(0.27844, 0.30104)**
FPGA SIMD Ver.	-0.52667	0.00439	**(-0.53797, -0.51537)**
Data Size			z= 2.576
500KB	-1.13587	0.00801	**(-1.15650, -1.11524)**
1MB	-0.94034	0.00801	**(-0.96097, -0.91971)**
2MB	-0.72871	0.00801	**(-0.74934, -0.70808)**
4MB	-0.52003	0.00801	**(-0.54066, -0.49939)**
8MB	-0.28391	0.00801	**(-0.30454, -0.26328)**
16MB	-0.06366	0.00801	**(-0.08429, -0.04303)**
32MB	0.18030	0.00801	**(0.15967, 0.20093)**
64MB	0.45237	0.00801	**(0.43174, 0.47300)**
128MB	0.73470	0.00801	**(0.71407, 0.75533)**
256MB	1.01595	0.00801	**(0.99532, 1.03659)**
512MB	1.28919	0.00801	**(1.26856, 1.30982)**

Table 5: Confidence levels for the effects in the data transform experiment

Confidence Intervals for Interactions

Std Deviation 0.013872 z= 2.57600

	CPU Seq.	CPU OMP	FPGA Naïve	FPGA SIMD
500KB	**(-0.26892, -0.19745)**	**(-0.22616, -0.15469)**	**(-0.08389, -0.01243)**	**(0.43604, 0.50750)**
1MB	**(-0.16860, -0.09713)**	**(-0.19231, -0.12084)**	**(-0.13392, -0.06245)**	**(0.35189, 0.42336)**
2MB	**(-0.12732, -0.05585)**	(-0.06409, 0.00738)	**(-0.14993, -0.07846)**	**(0.19841, 0.26987)**
4MB	(-0.04318, 0.02829)	**(0.00496, 0.07643)**	**(-0.12850, -0.05703)**	**(0.02379, 0.09525)**
8MB	**(0.04396, 0.11543)**	(-0.01643, 0.05503)	**(-0.10066, -0.02920)**	(-0.06979, 0.00167)
16MB	**(0.05634, 0.12781)**	**(0.00901, 0.08048)**	(-0.03828, 0.03318)	**(-0.17000, -0.09854)**
32MB	**(0.02140, 0.09287)**	**(0.03007, 0.10154)**	**(0.00751, 0.07898)**	**(-0.20193, -0.13046)**
64MB	(-0.00637, 0.06510)	**(0.06857, 0.14004)**	**(0.03082, 0.10229)**	**(-0.23596, -0.16449)**
128MB	**(0.01108, 0.08255)**	**(0.04733, 0.11880)**	**(0.04663, 0.11810)**	**(-0.24798, -0.17651)**
256MB	**(0.03054, 0.10200)**	**(0.01054, 0.08201)**	**(0.06500, 0.13647)**	**(-0.24901, -0.17755)**
512MB	**(0.05799, 0.12946)**	(-0.06456, 0.00691)	**(0.09214, 0.16361)**	**(-0.22852, -0.15705)**

Figure 6: Confidence levels for the interactions in the data transform experiment

The confidence intervals of the effects can be seen in tables 5 and 6 and are bold if they are significant. The data size and processing platform are all significant, with a z value of 2.576 resulting in a 99% confidence in the claim (Jain). While not all interactions between effects are confident at this level a large portion of them are which support the earlier claim.

Figure 3: A quantile-quantile and Residuals vs Predicted plot for data transformation values

As a final analysis a plot of the residuals vs the predicted values is shown in figure 3 along with a quantile-quantile plot of the data. In the residual plot there does not appear to be a trend with the residuals with a small axis step size, which makes an argument for the assumption of independence between the runs. With the quantile-quantile plot the overall may appear fairly linear at first the tails at the upper and lower quantile along with a heavy amount of points sitting near the origin may point to the errors in this data to not be normal. However the scale of the errors is small and account for a small portion of the variation compared to the major effects. This may point to a distribution that is peakier that normal.

4.1. System Execution

Log of System Execution Time in Miliseconds (Averaged from Three Replications)

	CPU Seq.	CPU OMP	FPGA Naïve	FPGA SIMD	Row Sum	Row Avg.	Row Effect
500KB	0.46394	0.14778	3.79025	3.79040	8.19236	2.04809	-0.63960
1MB	0.72498	0.34226	3.78398	3.78760	8.63882	2.15970	-0.52799
2MB	0.97478	0.60987	3.79755	3.78193	9.16413	2.29103	-0.39666
4MB	1.23491	0.81434	3.79652	3.78870	9.63446	2.40861	-0.27908
8MB	1.50661	1.05829	3.78477	3.77922	10.12890	2.53222	-0.15547
16MB	1.76474	1.31579	3.79219	3.79695	10.66966	2.66742	-0.02028
32MB	2.00141	1.59754	3.79534	3.79515	11.18944	2.79736	0.10967
64MB	2.24588	1.90578	3.81273	3.79667	11.76106	2.94026	0.25257
128MB	2.54496	2.19669	3.84049	3.78900	12.37113	3.09278	0.40509
256MB	2.84665	2.45670	3.87873	3.80432	12.98639	3.24660	0.55891
512MB	3.14628	2.68589	3.88041	3.80953	13.52211	3.38053	0.69283
Col. Sum	19.45512	15.13091	41.95297	41.71946	118.25846		
Col. Avg.	1.76865	1.37554	3.81391	3.79268		2.68769	
Col. Effect	-0.91905	-1.31216	1.12621	1.10499			

Interactions Between Execution Platform and Data Size

	CPU Seq.	CPU OMP	FPGA Naïve	FPGA SIMD
500KB	-0.66864	-0.58824	0.61775	0.63913
1MB	-0.51921	-0.50537	0.49987	0.52472
2MB	-0.40056	-0.36961	0.38229	0.38789
4MB	-0.25819	-0.28220	0.26349	0.27690
8MB	-0.08193	-0.17125	0.11875	0.13443
16MB	0.02868	-0.04483	-0.00492	0.02107
32MB	0.11956	0.11225	-0.12643	-0.10539
64MB	0.22113	0.27758	-0.25194	-0.24678
128MB	0.36587	0.42145	-0.37853	-0.40879
256MB	0.51202	0.53280	-0.49582	-0.54900
512MB	0.68127	0.61743	-0.62452	-0.67418

Figure 3: The effects and interactions table for the System Execution

In table 7 the computation of effects for the system execution are shown. Here in contrast to the data transformation we see an increased execution time of 2.69 log milliseconds (489.77 ms) which would make sense as the timer is capturing more time. The effects due to data size are closer by comparison to the data transform (~1.2 log milliseconds) and the effects due to the processing platform are close to 2.4 log milliseconds away from each other in an FPGA vs CPU case. Interestingly enough, the FPGA implementations all have roughly the same execution time of about 3.7-3.8 log milliseconds which seems to denote that the overhead of OpenCL is overshadowing the data transform itself.

ANOVA Table						
Componet	SSQ	Percentage of	Degrees of freedom	Mean Square	F-Compute	F-Table
y…	1166.03013		132			
ȳ...	953.52709		1			
y...-ȳ...	212.50304	100.00%	131			
A	166.83967	78.51%	3	55.61322	99939.67065	2.72
B	23.75976	11.18%	10	2.37598	4269.74490	1.94
AB	21.85464	10.28%	30	0.72849	1309.12872	1.59
error	0.04897	0.02%	88	0.00056		

Figure 3: ANOVA table for System Execution

Moving on to the ANOVA table (8), the most interesting part about this is that the processing platform accounts for 78.1% of the variation, a major step up from the previous measurement. Also here the dataset size and the interactions play less of a role in variation each taking up about 21.46% of the variation. Also, once again the errors are small only accounting for .02% of the total variation.

Confidence Intervals for Interactions (at 95% confidence)

Std Deviation 0.011246 z= 2.576

	CPU Seq.	CPU OMP	FPGA Naïve	FPGA SIMD
500KB	**(-0.69408, -0.63614)**	**(-0.61713, -0.55919)**	**(0.58698, 0.64492)**	**(0.60835, 0.66629)**
1MB	**(-0.54465, -0.48671)**	**(-0.53426, -0.47632)**	**(0.46909, 0.52703)**	**(0.49394, 0.55188)**
2MB	**(-0.42618, -0.36824)**	**(-0.39797, -0.34003)**	**(0.35134, 0.40928)**	**(0.35694, 0.41488)**
4MB	**(-0.28363, -0.22569)**	**(-0.31109, -0.25315)**	**(0.23272, 0.290660**	**(0.24613, 0.30406)**
8MB	**(-0.13554, -0.0776)**	**(-0.19075, -0.13281)**	**(0.09736, 0.15530)**	**(0.11305, 0.17098)**
16MB	(-0.01260, 0.04534)	**(-0.06844, -0.01050)**	(-0.03041, 0.02753)	(-0.00442, 0.05352)
32MB	**(0.09412, 0.15206)**	**(0.08336, 0.14130)**	**(-0.15720, -0.09926)**	**(-0.13616, -0.07823)**
64MB	**(0.19569, 0.25363)**	**(0.24870, 0.30664)**	**(-0.28271, -0.22477)**	**(-0.27755, -0.21961)**
128MB	**(0.34225, 0.40019)**	**(0.38710, 0.44503)**	**(-0.40748, -0.34954)**	**(-0.43774, -0.37980)**
256MB	**(0.49012, 0.54806)**	**(0.49328, 0.55122)**	**(-0.52305, -0.46511)**	**(-0.57623, -0.51829)**
512MB	**(0.65583, 0.71377)**	**(0.58854, 0.64648)**	**(-0.65530, -0.59736)**	**(-0.70496, -0.64702)**

Table 5: Confidence levels for the effects in the system execution

Confidence intervals for the Effects of Data Size and Platform (at 99% confidence)

Parameter	Mean Effect	Std Deviation	Confidence Interval
μ	2.68769	0.02359	
Processing Platform			z= 2.576
CPU Sequential Exec.	-0.91552	0.00356	**(-0.92468, -0.90636)**
CPU OpenMP Exec.	-1.31207	0.00356	**(-1.32123, -1.30291)**
FPGA Naïve Ver.	1.12441	0.00356	**(1.11525, 1.13357)**
FPGA SIMD Ver.	1.10318	0.00356	**(1.09402, 1.11234)**
Data Size			z= 2.576
500KB	-0.63960	0.00649	**(-0.65633, -0.62288)**
1MB	-0.52799	0.00649	**(-0.54471, -0.51126)**
2MB	-0.39666	0.00649	**(-0.41338, -0.37993)**
4MB	-0.27908	0.00649	**(-0.29580, -0.26235)**
8MB	-0.15547	0.00649	**(-0.17219, -0.13874)**
16MB	-0.02028	0.00649	**(-0.03700, -0.00355)**
32MB	0.10967	0.00649	**(0.09294, 0.12639)**
64MB	0.25257	0.00649	**(0.23585, 0.26930)**
128MB	0.40509	0.00649	**(0.38837, 0.42182)**
256MB	0.55891	0.00649	**(0.54218, 0.57563)**
512MB	0.69283	0.00649	**(0.67611, 0.70956)**

Figure 6: Confidence levels for the interactions in the system execution

The confidence intervals for the effects are listed in tables 9 and 10 and are bolded when they denote significance. Here, once again, the confidence interval for all the processing platforms and all the data set sizes are significant. A majority of the interactions are significant as well so the range that is listed can be considered correct.

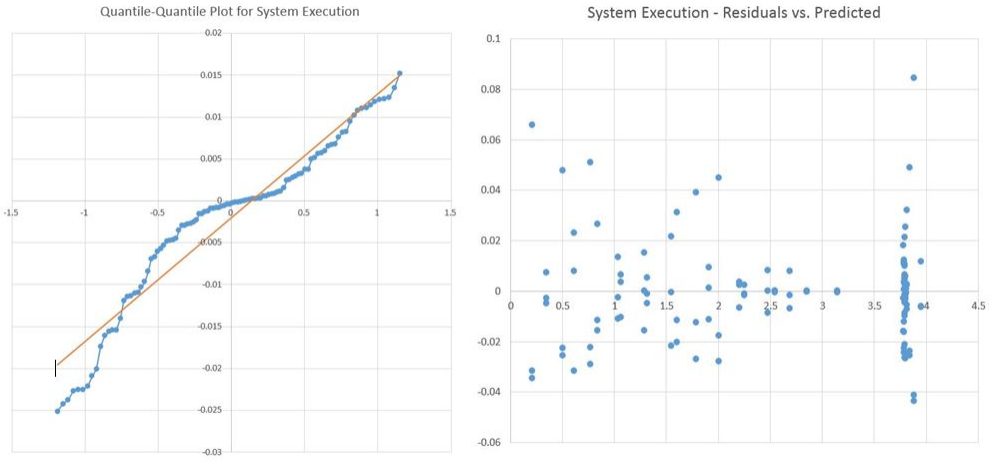

Figure 4: A quantile-quantile and Residuals vs Predicted plot for system execution values

Finally to test the model assumptions the residuals vs the predicted values are shown in figure 4. Here also, there is no visible trend in the residuals and the expected values. There is a fair amount of clustering around the 3.5 to 4 in the expected values range but this is happening because of the log transform as the points in that region all come from FPGA runs. In that region the execution time fall somewhere between 6000 milliseconds and 7500 milliseconds which is not easy to see when using the log scale. Moving onto the quantile-quantile plot, this time, a more normal distribution can be observed in the plot so we can more comfortably say that the errors are normally distributed.

4.3 Overall Execution Graphs

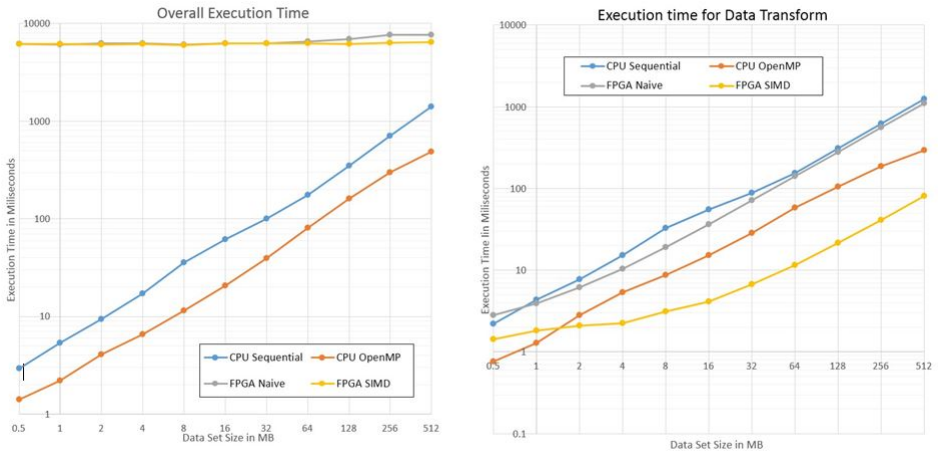

Figure 5: Average execution of each execution mode (shown in milliseconds)

The graphs shown here (figure 5) display the execution time of the program either by data transform or system execution. Each colored line represents a type of processing platform and the values are graphed without the log transformation but are plotted on a log-log graph. For the data transform OpenMP initially starts out in the lead as far as execution time but is soon defeated at a problem size of 2MB by the FPGA implementation. Interestingly the FPGA naive implementation did not perform too differently from a sequential CPU version of the program. In system execution however, the overhead of OpenCL rears its ugly head and one finally gets a taste of how much overhead is incurred by starting up and using the OpenCL libraries. Here, the FPGAs remain static at around 6000 milliseconds do not appear to deviate from that line. Of course it is still growing but the performance is hard to see on a log scale like this where incremental gain is a small fraction of the total execution. Combined with the data collected and analyzed in the subsections above these two graphs in conjunction paint an interesting picture. Overall, at 512MB we see the FPGA SIMD kernel has a 3.14x performance gain when looking at data transform over the OpenMP implementation but it is 13x slower when it comes to total system execution.

4.4 Discussion of the Results

The goal of this project was to try and evaluate if any performance gain could be achieved by using the HARP architecture to speed up simple data transformations in a stream processing paradigm. When looking at the results of just the data transform I think a resounding claim of yes can be made, however this comes with the caveat of performance can only be had if using large data sets for these types of problems otherwise OpenMP or another multi-threaded program may be faster at splitting up work between cores. When looking at system execution time the story takes an about face where the OpenCL programs are overwhelmed with their overhead and their data transformation only takes a fraction of the total time. Now of course this is a very general claim and may change depending on the application. If using paradigms like OpenCL pipes [

Intel/Altera '14] to pass data between kernels of execution using OpenCL for FPGA might be a wise choice as one would incur the cost of overhead in their application.

Also, the complexity in programming OpenCL is much easier than authoring kernels in VHDL but it is not easier than say adding in two lines to an existing C program which can be done using OpenMP. In programming the openCL app an additional 200 lines of code was used to set up the context, call the FPGA executable, setup buffers and enqueue the kernel for execution. Not to mention the synthesis time of the kernels is quite long sometimes taking up to 5 hours to have a complete synthesis. Like most things in the computing field there are benefits and drawbacks to every choice made in implementation. While 3.14x performance gain is excellent to have, the overhead of the software needs to be kept in mind when designing a system or application.

5 Conclusion

This paper set out to evaluate a stream processing data transformation using the Intel HARP system and OpenCL HLS. In the tests a simple fixed point to floating point conversion was used as the test application and two measurements were taken, the time spent doing the data transformation and the time spent running the application including overhead from the libraries used. Four different processing platforms were used to evaluate the implementation: a sequential CPU execution, an OpenMP execution, a naive OpenCL FPGA implementation and a SIMD OpenCL FPGA implementation. Another factor that was chosen was the amount of data to ingest and transform which varied from 512 KB to 512MB. The experimental design became a full factorial design with replications used to average the runs over three trials. The effects of each factor was proven to be significant and given the different set of measurements which factor had a greater hand in the variation of the measurements. In general an OpenCL FPGA with SIMD attributes was 3.14 times faster than the OpenMP application. When focusing on just the data transformation, however, the OpenCL libraries incur a large overhead by comparison. While this does not discount using the HARP and OpenCL for stream processing applications by any means a programmer must be aware of what would be a good application to target.

Future work down this path involves trying to figure out what improvements can be made to the system using possible all possible attribute settings, and authoring more complex stream processing applications to discover a few of the corner cases of the HARP architecture.

A1. List of References

1. [Zahran'16] Zahran, Mohamed, "Heterogeneous Computing: Hardware and Software Perspectives."(Applicative 2016). ACM, New York, NY, USA https://www.youtube.com/watch?v=H1lkkrt13v0
2. [Hussain '14] Hussain, T., Palomar O., Unsal,O., et.al "Advanced Pattern based Memory Controller for FPGA based HPC applications," 2014 International Conference on High Performance Computing & Simulation (HPCS), Bologna, 2014, pp. 287-294. http://ieeexplore.ieee.org/document/6903697/

3. [Intel/Altera '17] Intel/Altera, "Intel FPGA SDK for OpenCL Best Practices Guide," https://www.altera.com/documentation/mwh1391807516407.html [Reference guide for programming specifically using Altera's OpenCL compiler]
4. [Beard '17] Beard, Johnathan, "A SHORT INTRO TO STREAM PROCESSING," http://www.jonathanbeard.io/blog/2015/09/19/streaming-and-dataflow.html [Reference and introduction into stream processing.]
5. [Stone '10] Stone, J.E. , Gohara D. and Shi G., "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," in Computing in Science & Engineering, vol. 12, no. 3, pp. 66-73, May-June 2010. http://ieeexplore.ieee.org/document/5457293/
6. [Khronos '17] Khronos Group, "OpenCL 2.0 Reference Pages," https://www.khronos.org/registry/OpenCL/sdk/2.0/docs/man/xhtml/ [Reference documents for OpenCL libraries by the creators of the OpenCL specification]
7. [Putnam '14] Putnam, Andrew , Caulfield, Adrian M. , et. al "A reconfigurable fabric for accelerating large-scale datacenter services" In Proceeding of the 41st annual international symposium on Computer architecture (ISCA 2014), June 2014, pp. 13-24. https://www.microsoft.com/en-us/research/publication/a-reconfigurable-fabric-for-accelerating-large-scale-datacenter-services/
8. [Fletcher '11] Fletcher, Christopher , Lebedev, Ilia, et. al "Bridging the GPGPU-FPGA efficiency gap", In Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '11). February2011, pp119-122. http://dx.doi.org/10.1145/1950413.1950439
9. [Zohouri '16] Zohouri,Hamid Reza , Maruyama Naoya , et al. "Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs." In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis" (SC '16), Article 35 , 12 pages. https://dl.acm.org/citation.cfm?id=3014951
10. [Ndu '15] Ndu, Geoffrey , Navaridas,Geoffrey , and Lujan, Mikel. "CHO: towards a benchmark suite for OpenCL FPGA accelerators", In Proceedings of the 3rd International Workshop on OpenCL (IWOCL '15). Article 10 , 10 pages. http://dx.doi.org/10.1145/2791321.2791331
11. [Luo '17] Luo Y. et al., "Evaluating irregular memory access on OpenCL FPGA platforms: A case study with XSBench," 2017 27th International Conference on Field Programmable Logic and Applications (FPL), 2017, pp. 1-4. http://ieeexplore.ieee.org/document/8056827/
12. [Jain91] Jain, Raj, "The Art of Computer Systems Performance Analysis," John Wiley & Sons, INC, 1991, ISBN-10: 0471503363.
13. [Intel/Altera '14] Intel/Altera "OpenCL 2.0 Shared Virtual Memory Overview," https://software.intel.com/en-us/articles/opencl-20-shared-virtual-memory-overview [Reference guide for using SVM memory in OpenCL]

A2. List of Acronyms

ANOVA Analysis of Variance
CPU Central Processing Unit
FPGA Field Programmable Gate Array

GPU	Graphic Processing Unit
HARP	Hardware Accelerator Research Program
HDL	Hardware Description Language
HLS	High Level Synthesis
SIMD	Single Instruction Multiple Data
VHDL	Virtual Hardware Description Language

Last Modified: December 15, 2017
This and other papers on performance analysis of computer systems are available online at
http://www.cse.wustl.edu/~jain/cse567-17/index.html
Back to Raj Jain's Home Page