Neural Network Image Detection for SmartSSD Platform

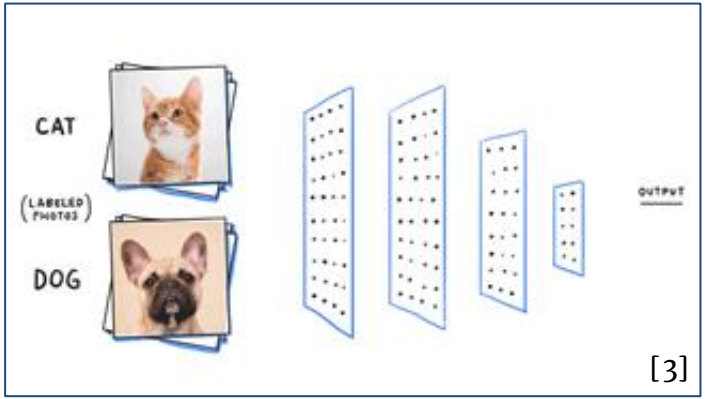
Jackson Hafele, William Zogg

CPR E 563 Final Report Spring 2023

IOWA STATE UNIVERSITY

Motivation

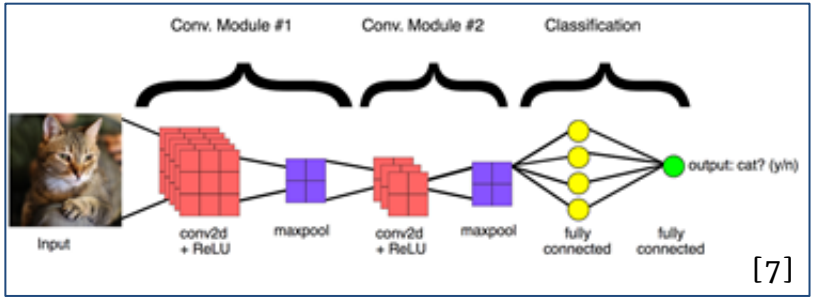
- Image detection is a notoriously high-BW ML application. [1]
- Computational Storage Devices (CSD) bring ML acceleration closer to the necessary storage. [2]
- Smart SSD can be an all-in-one solution combining:
  - Large, fast flash storage to store an entire neural network.
  - Close, customizable FPGA acceleration to utilize high BW.



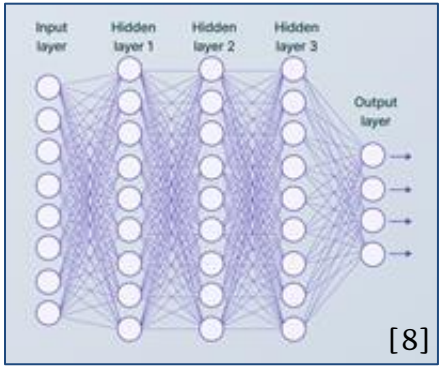
[3]

Background - Neural Network Application

- Image classification using the TinyImageNet dataset
- Utilizes 12 layer neural network to classify an image into 1 of 200 classes
- Each layer is computationally heavy.
- Thousands of data values loaded for every single inference



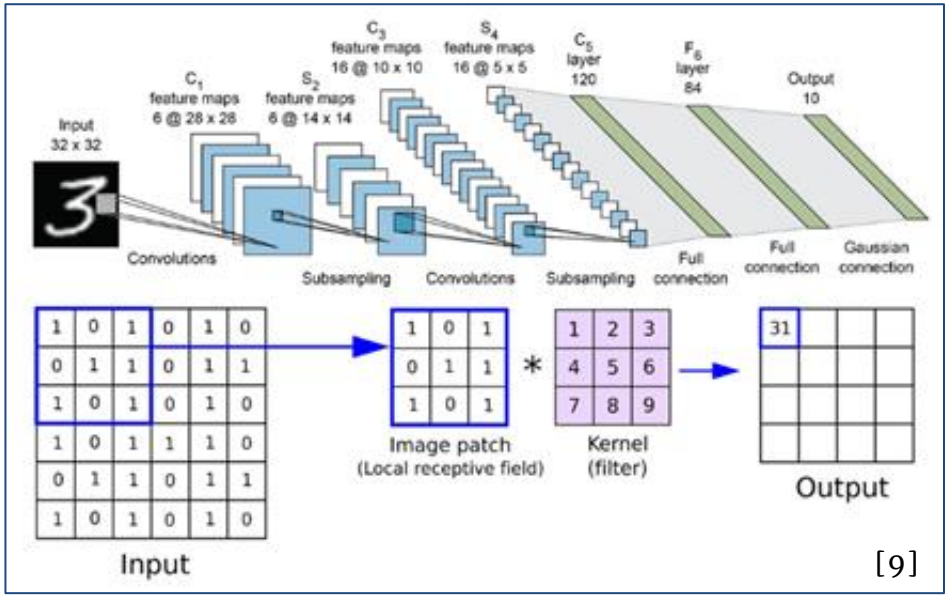
[7]



[8]

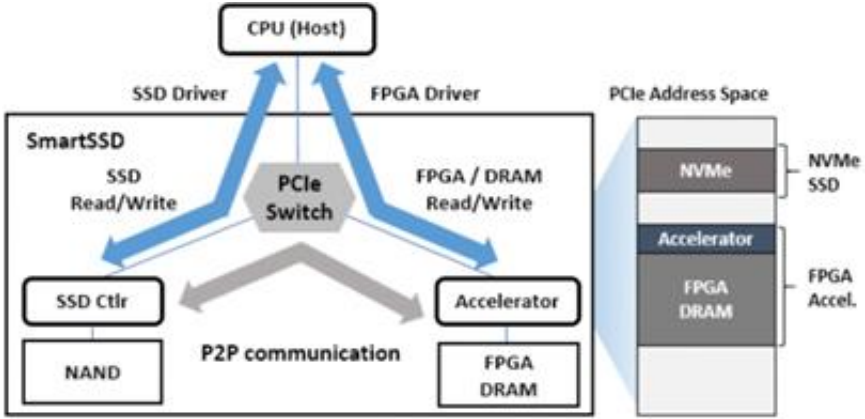
Background - Convolution Process

- Evaluate strength of a filter to identify features in images
- Higher number -> Higher presence of filter
- Data Used:
  - Input Data (3D array)
  - Filter Data (4D array)
  - Output Data (3D array)



Background - SmartSSD

- 3.84 TB NAND Flash Samsung SSD [4]
  - NVMe Connection
- AMD Kintex Ultrascale+ FPGA
  - Utilize Vitis to synthesize C
  - Reconfigurable
- PCIe Switch between SSD, FPGA, and Host CPU



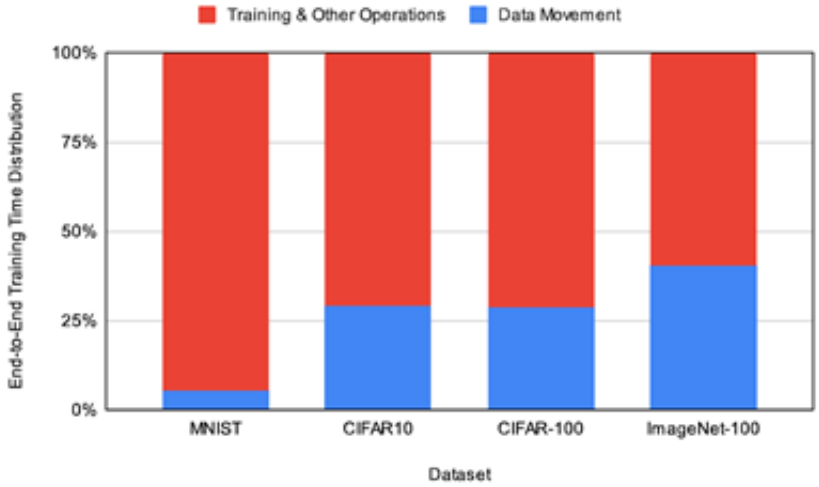
SmartSSD Architecture [5]

Related Works - NeSSA

- Data movement can take as much as 80% of total time for warehouse-scale operations
- Proposes near-storage acceleration model involving SmartSSD to reduce data movement
- Select subsets of large datasets to train while maintaining accuracy
  - Uses multiple datasets to train model (includeTinyImageNet)
- Analyzes accuracy of NeSSA training and time to train
  - Compares multiple subset sizes with other training models [6]

Related Works - NeSSA

- Authors compare time spent on computation and data movement for multiple models
- Demonstrates importance of improving Data Movement latency for future works



Percent of time Training Neural Networks Vs. Data Movement Latency for Machine Learning Models[6]

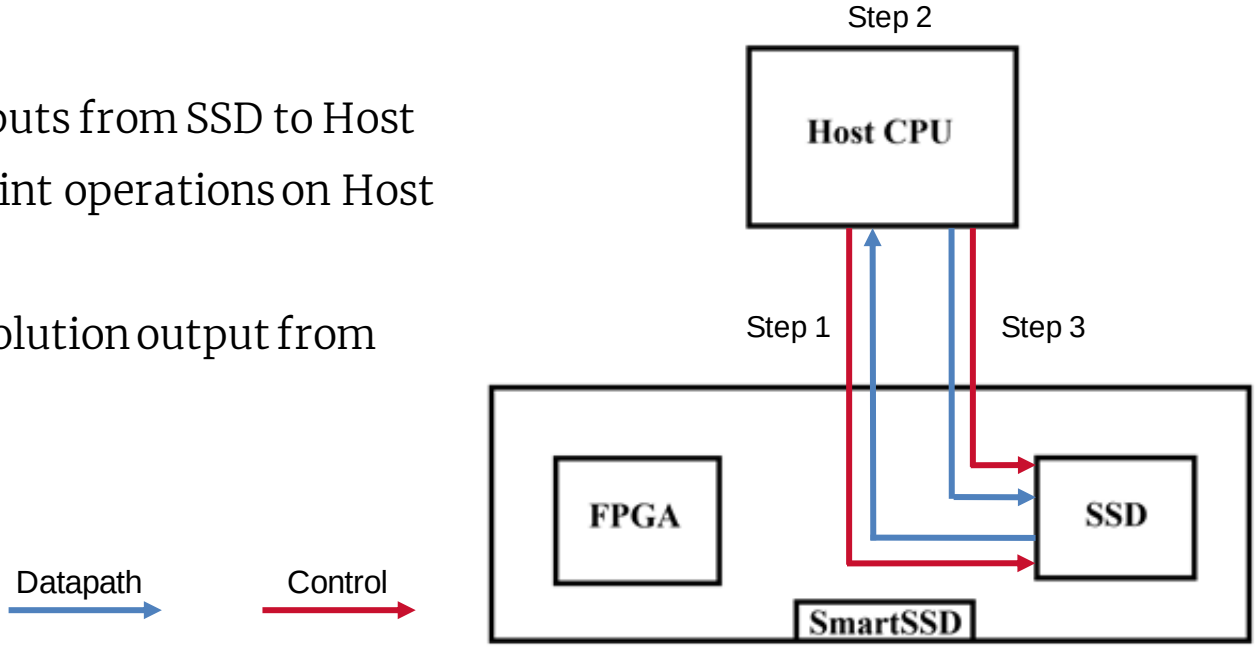
Solution Approach

1. Run Neural Network application on Host Machine
    a. Utilize existing CPRE487/587X library, gain baseline results
2. Convolution Kernel Synthesis
    a. Refactor Convolution Layer to individual function
    b. Synthesize Kernel in Vitis
3. Run Neural Network application on SmartSSD application
    a. Compile with Vitis
    b. Peer to Peer communication between SSD and FPGA
4. Optimize Kernel design and memory transactions to reduce latency

Host Process

Host Code Steps

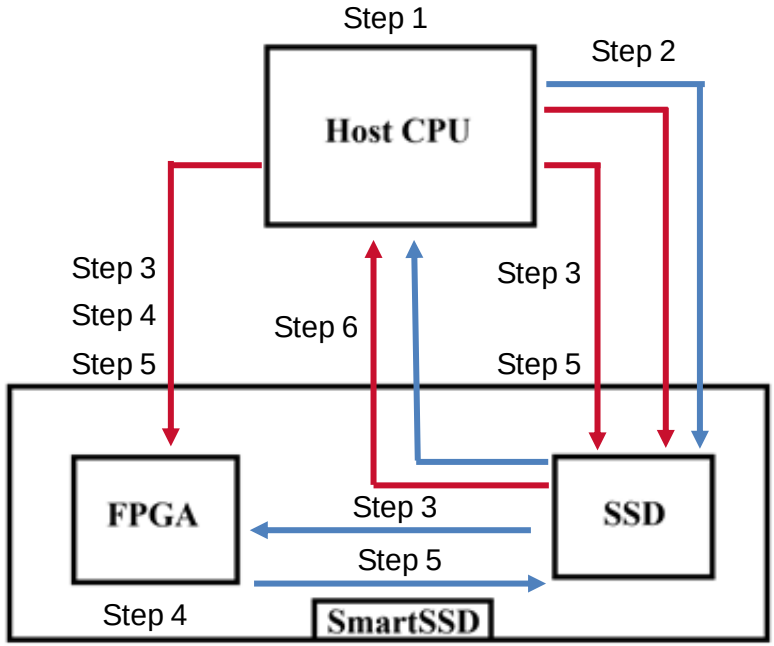
1. Read Convolution inputs from SSD to Host
2. Compute Floating Point operations on Host CPU
3. Write resulting convolution output from Host to SSD



SmartSSD Process

Host Code Steps

1. Initialize Kernel and SmartSSD
2. Write convolution inputs from Host to SSD in fixed blocks
3. Read input arguments from SSD to FPGA
4. Activate Kernel
5. Write output from FPGA to SSD
6. Read value from SSD to Host to Verify



Neural Network with Host

- Run baseline code on SmartSSD device WITHOUT FPGA

- **Majority of time spent on data movement**
  - Reading takes majority of time: 0.459 seconds
  - Writing data has minimal overhead: 0.002 seconds
- Computing FP operations takes 0.195 seconds
- A single inference for a low-res test image takes 0.672 seconds

- How can we reduce data movement with SmartSSD?

```
Running inference on layer 0
TIME: convolution layer took 0.0636671 seconds
Running inference on layer 1
TIME: convolution layer read took 0.459795 seconds
TIME: convolution layer operation took 0.195236 seconds
TIME: convolution layer write took 0.00251626 seconds
TIME: convolution layer took 0.672581 seconds
Running inference on layer 2
TIME: maxpooling layer took 0.000598622 seconds
Running inference on layer 3
TIME: convolution layer took 0.0779198 seconds
Running inference on layer 4
TIME: convolution layer took 0.136815 seconds
Running inference on layer 5
TIME: maxpooling layer took 0.000237458 seconds
Running inference on layer 6
TIME: convolution layer took 0.0248409 seconds
Running inference on layer 7
TIME: convolution layer took 0.0348511 seconds
Running inference on layer 8
TIME: maxpooling layer took 5.2217e-05 seconds
Running inference on layer 9
TIME: flatten layer took 1.0095e-05 seconds
Running inference on layer 10
TIME: dense layer took 0.00156071 seconds
Running inference on layer 11
TIME: dense layer took 0.000174442 seconds
Running inference on layer 12
TIME: softmax layer took 1.8034e-05 seconds
TIME: Model inference time 1.01346 seconds
```

Convolution Kernel

- Created separate class for Layer 2 Convolution
  - Enabled changes to Layer 2 while keeping rest of code same
  - Convolve operation moved to independent function
  - Called within the 3 level nested for loop

- Created function for single point convolution
  - Flattened 3D and 4D arrays into 1D arrays for synthesis and interfacing
  - 800 loop iterations to operate in kernel per call

Convolution Kernel

```
//Compute
for (int dataH = 0; dataH < 56; dataH++) { //dataHeight = 56
    for (int dataW = 0; dataW < 56; dataW++) { //dataWidth = 56
        for (int dataD = 0; dataD < 32; dataD++) { //dataDepth = 32

            count = 0;
            for (int weightD = 0; weightD < 32; weightD++) { //dataInDepth = 32
                for (int weightW = 0; weightW < 5; weightW++) { //layerWeightWidth = 5
                    for (int weightH = 0; weightH < 5; weightH++) { //layerWeightHeight = 5
                        dataIn_Internal[count] = dataIn_data[dataH + weightH][dataW + weightW][weightD];
                        layerWeight_Internal[count] = layerWeight_data[weightH][weightW][weightD][dataD];
                        count++;
                    }
                }
            }

            computePointHLS(dataIn_Internal, &dataOut_data[dataH][dataW][dataD], layerWeight_Internal, &layerBias_data[dataD]);

            dataOut_data[dataH][dataW][dataD] = std::max(fp32(0), dataOut_data[dataH][dataW][dataD]);

            printf("%f \n", dataOut_data[dataH][dataW][dataD]);
        }
    }
}
```

Layer 2 Convolution Function

Convolution Kernel

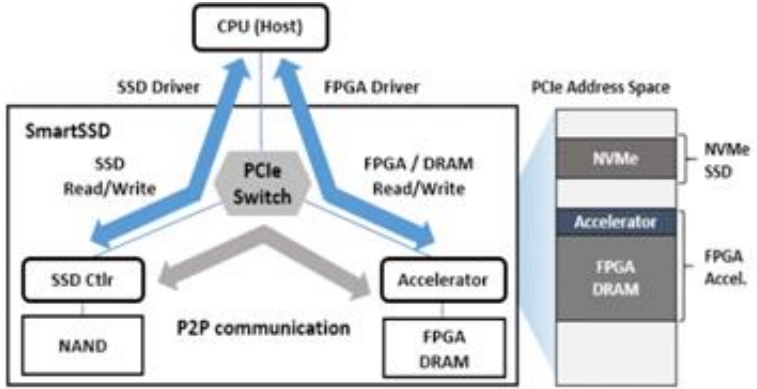
```
//Compute the convolution using threads
void computePointHLS(float dataIn_data[800], float* dataOut_data, float layerWeight_data[800], float* layerBias_data) {
#pragma HLS interface mode=m_axi  port=dataIn_data
#pragma HLS interface mode=s_axilite  port=dataOut_data
#pragma HLS interface mode=m_axi  port=layerWeight_data
#pragma HLS interface mode=s_axilite  port=layerBias_data

    float dataOut_Internal;

    //Compute
    for (int i = 0; i < 32*5*5; i++) { //dataInDepth = 32
        dataOut_Internal += dataIn_data[i] * layerWeight_data[i];
    }

    dataOut_Internal += *layerBias_data;

    //Store dataOut_data
    *dataOut_data = dataOut_Internal;

}
```

Single Point Convolution Function, Target Kernel

SmartSSD Application

- Created Host code for Peer to Peer memory transactions between Host/SSD and SSD/FPGA
- Synthesized Kernel code to run on SmartSSD FPGA

- Verified expected kernel output of convolution point
- Collected timing metrics with **chrono** C++ library
  - Inconsistent results for individual read/compute/write operations



SmartSSD Architecture [5]

SmartSSD Optimization

- Problem
  - Latency of FPGA kernel extremely slow
  - Memory transactions limited by 512 Byte minimum read/write size
  - Read/Write transactions not utilizing high bandwidth
- Solution
  - Reduce amount of empty data read/write due to minimum block size to increase data bandwidth
  - Reduce Kernel latency by adding parallelization of single point
  - Compute multiple points in 1 kernel call to increase data bandwidth
    - Less read/write transaction overhead from host

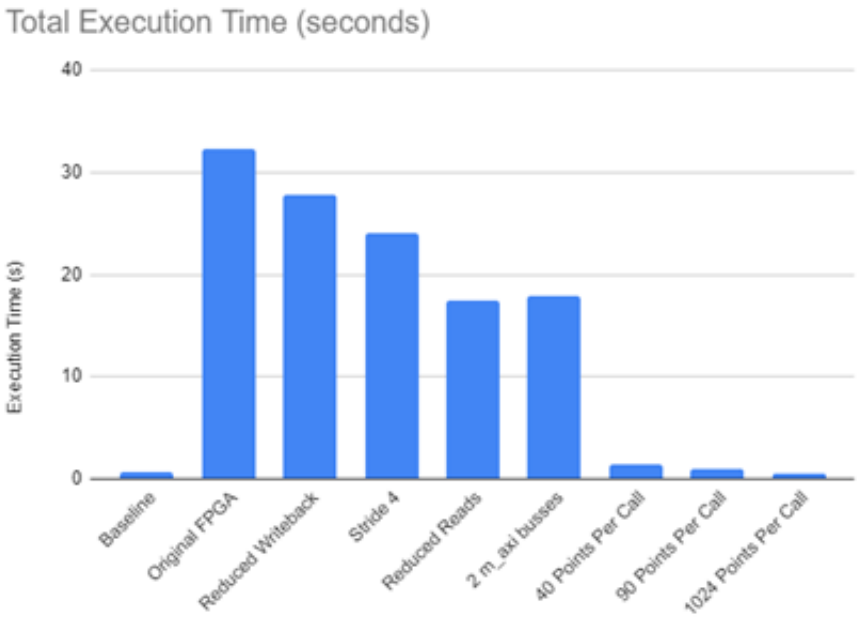
SmartSSD Optimization

- Kernel Improvements
  - Old Kernel: 9107 cycles for 1 point
  - Final Kernel: 708,747 cycles for 1024 points, average 692 cycles for 1 point
    - Computes 1024 points per kernel call, loop unrolling with stride 32
    - Average 0.865 cycles per loop iteration with parallelization
- Read Improvements
  - Old Host: Reads 4096 Values per point, Only 1602 Used
  - Final Host: Reads 1600 Values per point, Fully utilized
- Write Improvements
  - Old Host: Writes 1024 Values per point, only 1 Used
  - Final Host: Writes 1 value per point, Fully utilized

Results

- Full Layer Baseline CPU: 0.662 seconds
- Original SmartSSD: 32.246 seconds
- 48x SPEED DOWN

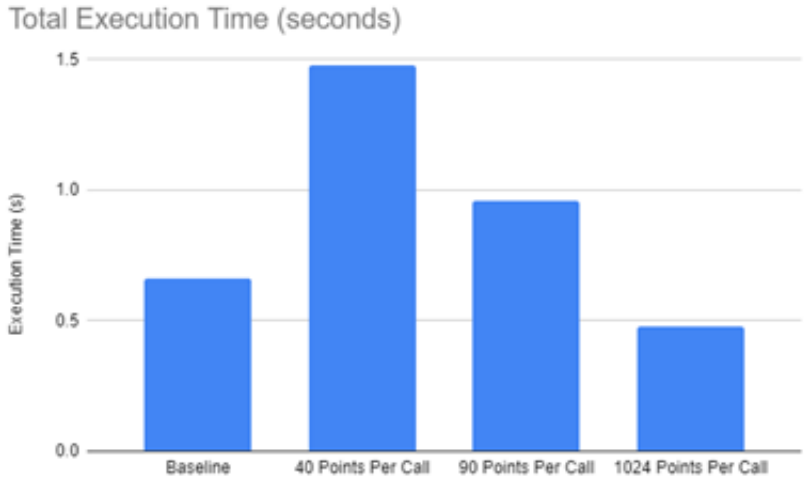
- Large improvement when running multiple convolution operations in one kernel
- Majority of time spent reading data from SSD and computing kernel



Results

- Full Layer Baseline CPU: 0.662 seconds
- 1024 Points per Kernel: 0.4778 seconds
- 1.38x SPEED UP

- More points per call ->
  - less latency overhead from kernel call control
  - Higher data bandwidth in read transactions



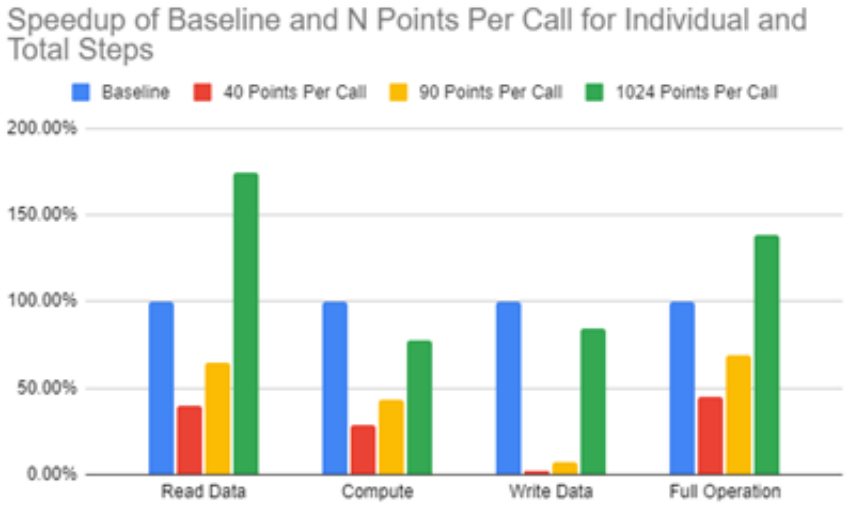
Results

- Speedup normalized to Baseline CPU
- Data read from SSD: 1.74X SPEED UP

- Kernel computations still slightly slower than baseline, room for improvement with Vitis HLS

- Data writes slower, due to low bandwidth transaction of 1 block writeback per 1024 points

	Baseline	40 Points Per Call	90 Points Per Call	1024 Points Per Call
Read Data	100.00%	39.98%	64.83%	174.87%
Compute	100.00%	28.67%	43.19%	77.91%
Write Data	100.00%	1.90%	7.47%	84.40%
Full Operation	100.00%	44.85%	69.17%	138.61%

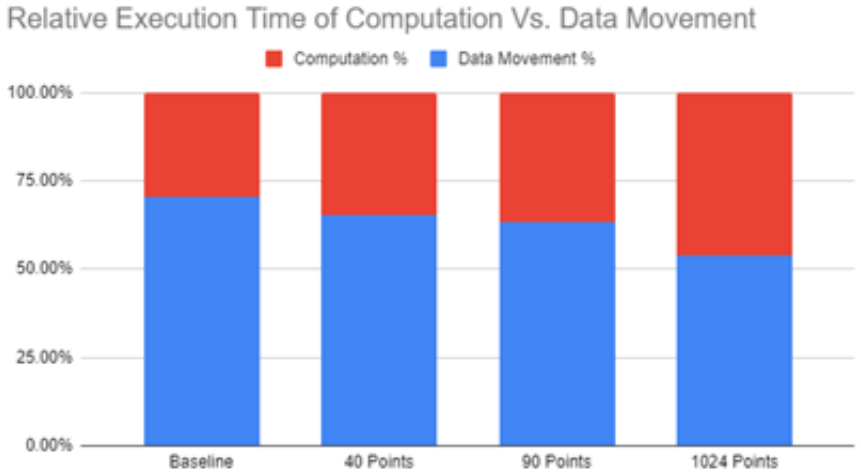


Results

- **Baseline CPU**: 70.50% moving data
- **1024 Point**: 53.82% moving data

- As data bandwidth increases per N point operations, % data movement decreases
- Final design also has lower data movement latency
- Leaves more room for improvement in optimizing kernel or write transactions

	Baseline	40 Points	90 Points	1024 Points
Data Movement %	70.50%	65.23%	63.54%	53.82%
Computation %	29.50%	34.77%	36.46%	46.18%

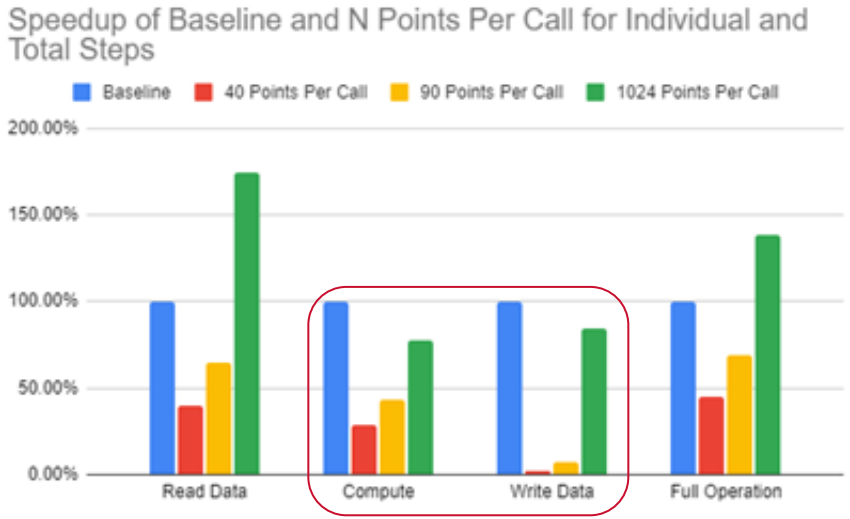


Analysis

- Application specific designs will take tailoring to achieve performance benefits, as seen with original FPGA design
- SmartSSD can provide speedup in terms of overall execution time while lowering the gap between data movement and computation time
- Designs benefit from high bandwidth applications and parallelization of FPGA design

Future Work

- Potential improvement for computation time of floating point operations in FPGA kernel
  - More parallelization, compute 1024 points faster
- Could speed up writeback by increasing write bandwidth larger than 1 4KB block
  - Store full layer of points then write back with one write transaction



References

[1] J. Do, V. C. Ferreira, H. Bobarshad, M. Torabzadehkashi, S. Rezaei, A. Heydarigorji, D. Souza, B. F. Goldstein, L. Santiago, M. S. Kim, P. M. V. Lima, F. M. G. França, and V. Alves, "Cost-effective, energy-efficient, and scalable storage computing for large-scale ai applications," ACM Trans. Storage, vol. 16, oct 2020.

[2] D. Fakhry, M. Abdelsalam, M. W. El-Kharashi, and M. Safar, "A review on computational storage devices and near memory computing for high performance applications," Memories - Materials, Devices, Circuits and Systems, vol. 4, p. 100051, 2023.

[3] https://miro.medium.com/max/700/1*oB3S5yHHhvougJkPXuc8og.gif

[4] https://www.xilinx.com/applications/data-center/computational-storage/smartssd.html

[5] J. H. Lee, H. Zhang, V. Lagrange, P. Krishnamoorthy, X. Zhao and Y. S. Ki, "SmartSSD: FPGA Accelerated Near-Storage Data Analytics on SSD," in *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 110-113, 1 July-Dec. 2020, doi: 10.1109/LCA.2020.3009347. keywords: {Field programmable gate arrays;Bandwidth;Random access memory;IP networks;Pipelines;Data analysis;Throughput;SmartSSD;data analytics;spark;parquet;SSD}

[6] Neha Prakriya, Yu Yang, Baharan Mirzasoleiman, Cho-Jui Hsieh, and Jason Cong. 2023. NeSSA: Near-Storage Data Selection for Accelerated Machine Learning Training. In Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '23). Association for Computing Machinery, New York, NY, USA, 8–15. https://doi.org/10.1145/3599691360340 4

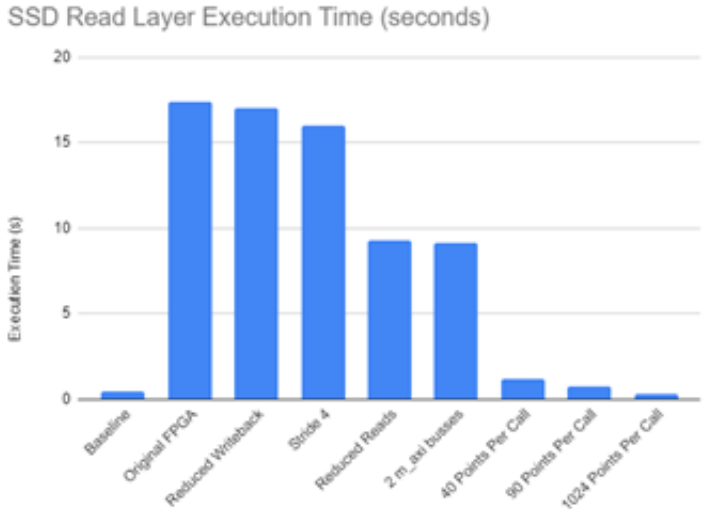
[7] https://miro.medium.com/v2/resize:fit:1200/1*QnKckNSZilG3HxytJZUoAw.png

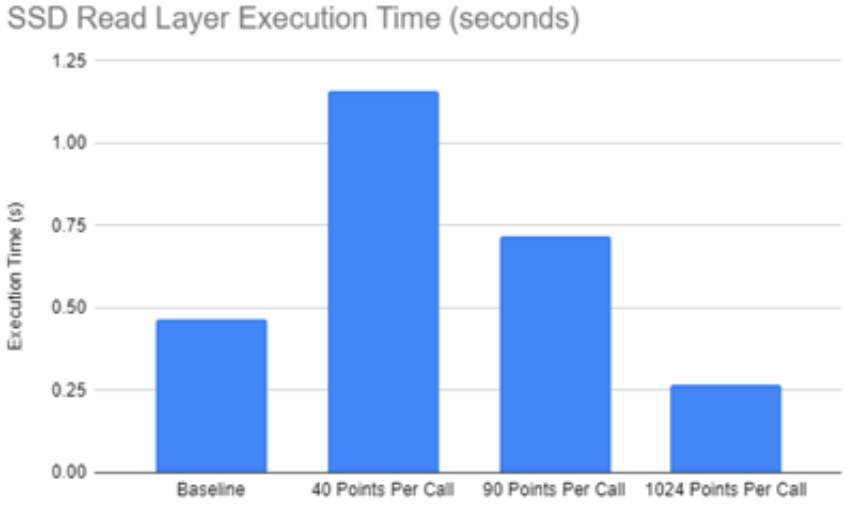
[8] https://assets-global.website-files.com/5d7b77b063a9066d83e1209c/627d1225cb1b3d197840427a_60f040a887535b932a3b2b6e_cnn-hero%2520(1).png

[9] https://assets-global.website-files.com/614c82ed388d53640613982e/646371e3bdc5ca90dee5331b_convolutional-neural-network%20(1).webp

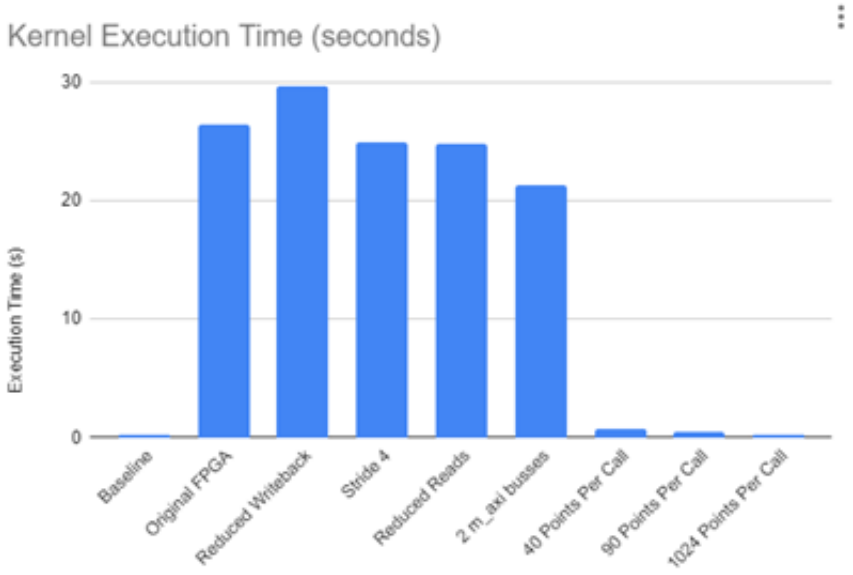
Questions

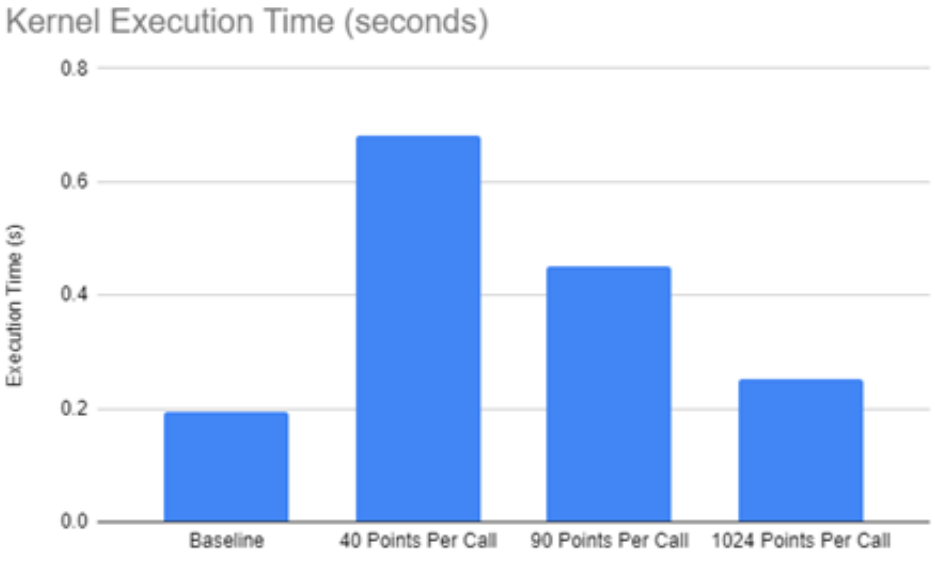
Supplemental



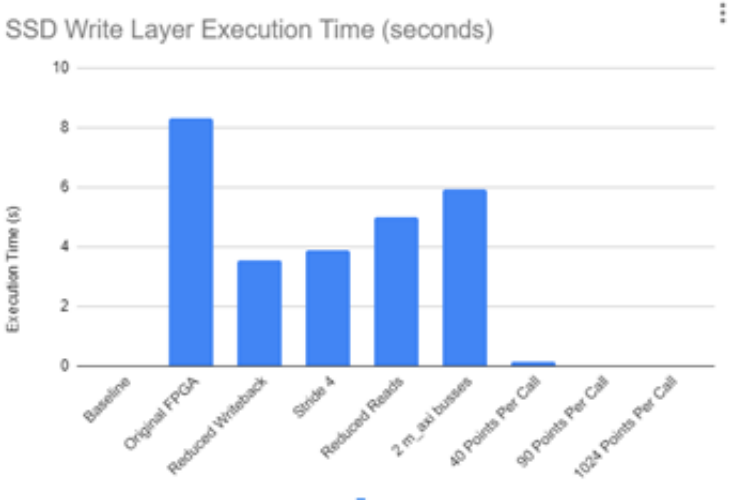


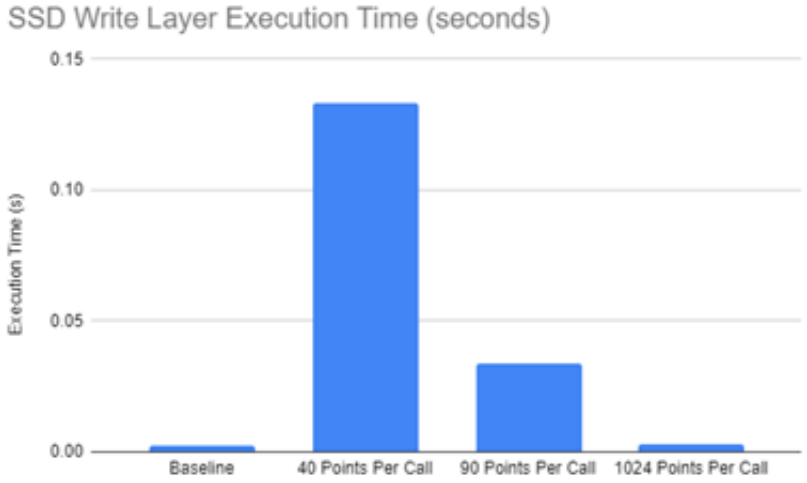
Supplemental





Supplemental





Supplemental

	Baseline	Original FPGA	Reduced Write	Stride 4	Reduced Read	2 m_axi busses	40 Points	90 Points	1024 Points
Point Time (us)	35.3864	173.0000	169.4000	159.4000	92.6000	91.0000	11.5750	7.1378	2.6463
Layer Time (s)	0.4644	17.3609	16.9996	15.9961	9.2926	9.1320	1.1616	0.7163	0.2656
Layer Difference (s)	0.0000	-16.8965	-16.5352	-15.5317	-8.8282	-8.6676	-0.6972	-0.2519	0.1988
Speedup	1.0000	0.0267	0.0273	0.0290	0.0500	0.0509	0.3998	0.6483	1.7487

Table II: SSD Read Latency

	Baseline	Original FPGA	Reduced Write	Stride 4	Reduced Read	2 m_axi busses	40 Points	90 Points	1024 Points
Point Time (us)	2.5044	263.6000	295.4000	248.2000	246.8000	211.8000	6.7900	4.5067	2.4986
Layer Time (s)	0.1953	26.4528	29.6440	24.9074	24.7669	21.2546	0.6814	0.4523	0.2507
Layer Difference (s)	0.0000	-26.2574	-29.4486	-24.7120	-24.5715	-21.0592	-0.4860	-0.2569	-0.0554
Speedup	1.0000	0.0074	0.0066	0.0078	0.0079	0.0092	0.2867	0.4319	0.7791

Table III: Single Point Kernel Computation Latency

Supplemental

	Baseline	Original FPGA	Reduced Write	Stride 4	Reduced Read	2 m_axi busses	40 Points	90 Points	1024 Points
Point Time (us)	0.0590	82.8000	35.2000	38.6000	49.8000	59.2000	1.3300	0.3378	0.0299
Layer Time (s)	0.0025	8.3091	3.5324	3.8736	4.9975	5.9408	0.1335	0.0339	0.0030
Layer Difference (s)	0.0000	-8.3066	-3.5299	-3.8711	-4.9950	-5.9383	-0.1309	-0.0314	-0.0005
Speedup	1.0000	0.0003	0.0007	0.0007	0.0005	0.0004	0.0190	0.0747	0.8440

Table IV: SSD Write Latency

	Baseline	Original FPGA	Reduced Write	Stride 4	Reduced Read	2 m_axi busses	40 Points	90 Points	1024 Points
Point Time (us)	6.5994	321.3449	277.1923	240.3659	174.4021	179.0099	14.7142	9.5404	4.7612
Layer Time (s)	0.6623	32.2476	27.8168	24.1212	17.5016	17.9640	1.4766	0.9574	0.4778
Layer Difference (s)	0.0000	-31.5853	-27.1545	-23.4589	-16.8393	-17.3017	-0.8143	-0.2951	0.1845
Layer Speedup %	1.0000	0.0205	0.0238	0.0275	0.0378	0.0369	0.4485	0.6917	1.3861

Table V: Total Single Point Latency

Supplemental

Latency Information

Compute Unit	Kernel Name	Module Name	Start Interval	Best (cycles)	Avg (cycles)	Worst (cycles)	Best (absolute)	Avg (absolute)	Worst (absolute)
computePointHLS_1	computePointHLS	computePointHLS_Pipeline_VITIS_LOOP_17_1	8883	8883	8883	8883	29.607 us	29.607 us	29.607 us
computePointHLS_1	computePointHLS	computePointHLS	9108	9107	9107	9107	30.354 us	30.354 us	30.354 us

Original Kernel Latency Delays

Latency Information

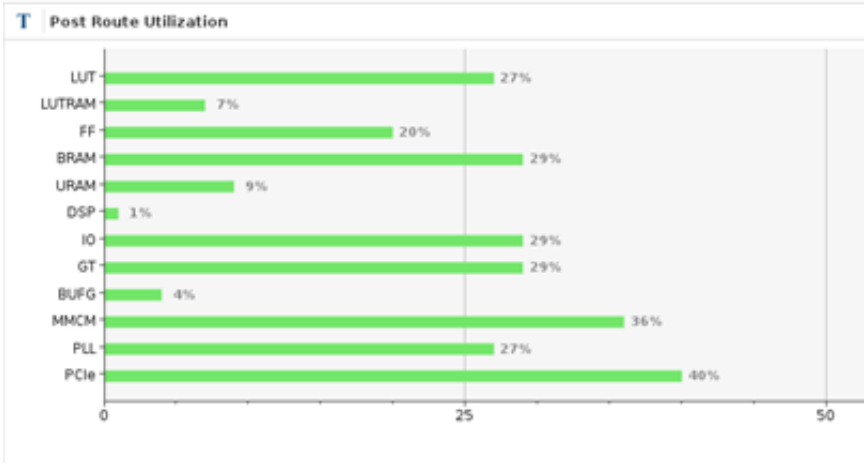
Compute Unit	Kernel Name	Module Name	Start Interval	Best (cycles)	Avg (cycles)	Worst (cycles)	Best (absolute)	Avg (absolute)	Worst (absolute)
computePointHLS_1	computePointHLS	computePointHLS_Pipeline_VITIS_LOOP_104_2	34	34	34	34	0.113 us	0.113 us	0.113 us
computePointHLS_1	computePointHLS	computePointHLS_Pipeline_VITIS_LOOP_109_3	296	296	296	296	0.987 us	0.987 us	0.987 us
computePointHLS_1	computePointHLS	computePointHLS	708748	708747	708747	708747	2.362 ms	2.362 ms	2.362 ms

Final Kernel Latency Delays

Supplemental

Kernel Synthesis Utilization

Name	LUT	LUTAsMem	REG	BRAM	URAM	DSP
Platform	147967	9990	205724	257	12	9
User Budget	374753	151290	839716	727	116	1959
Used Resources	9715	1199	13127	24	0	15
Unused Resources	365038	150091	826589	703	116	1944
computePointHLS (1)	9715	1199	13127	24	0	15
computePointHLS_1	9715	1199	13127	24	0	15



Final Kernel Utilization

Supplemental

```
void computePointHLS(float* dataIn, float* layerWeight, float* dataOut_final) {
#pragma HLS interface mode=m_axi      port=dataIn         bundle=gmem0
#pragma HLS interface mode=m_axi      port=layerWeight    bundle=gmem1
#pragma HLS interface mode=m_axi      port=dataOut_final  bundle=gmem0
float temp_add[STRIDE];

for(int point = 0; point < NUM_POINTS; point++){

    for(int i = 0; i < STRIDE; i++){ //Reset Temp Values
            temp_add[i] = 0;
    }

    for (int i = 0; i < POINT_SIZE; i += STRIDE) { //Compute
        #pragma HLS pipeline

        for(int j=0; j<STRIDE; j++){ //Loop Unrolling
            temp_add[j] += dataIn[POINT_SIZE*point + i+j] * layerWeight[POINT_SIZE*point + i+j];
        }
    }

    for(int i=1; i<STRIDE; i++){ //Sum temp values
    #pragma HLS unroll
        temp_add[0] += temp_add[i];
    }

    dataOut_final[point] = temp_add[0]; //Store dataOut_data
```

Final Kernel Code

Supplemental

```
if (pwrite(nvmeFd, static_cast<void*>(in_data.data()), vector_size_bytes_write, BLOCK_SIZE_BYTES * IN_OFFSET) < 0) { //Error if <0
        printf("WRITE FAIL! %s\n", strerror(errno));
        exit(EXIT_FAILURE);
}
```

Write to SSD From Host

```
if (pread(nvmeFd, static_cast<void*>(out_final.data()), vector_size_bytes_read, BLOCK_SIZE_BYTES * OUT_FINAL_OFFSET) < 0) { //Error if <0
        printf("READ FAIL! %s\n", strerror(errno));
        exit(EXIT_FAILURE);
}
```

Read from SSD to Host

Supplemental

```
if (pread(nvmeFd, (void*)p2pPtrRead_in_data, vector_size_bytes_write, BLOCK_SIZE_BYTES * IN_OFFSET) <= 0) {
    std::cerr << "ERR: pread failed: "
              << " error: " << strerror(errno) << std::endl;
    exit(EXIT_FAILURE);
}
```

1. Read from SSD to FPGA

```
// Set the Kernel Arguments
OCL_CHECK(err, err = krnl_computePointHLS.setArg(0, buffer_input_p2p_read_in_data));
OCL_CHECK(err, err = krnl_computePointHLS.setArg(1, buffer_input_p2p_read_in_weight));
OCL_CHECK(err, err = krnl_computePointHLS.setArg(2, buffer_output_p2p_write_out_final));
```

2. Set Input Arguments for Kernel to buffer pointers

```
OCL_CHECK(err, err = q.enqueueTask(krnl_computePointHLS));
OCL_CHECK(err, err = q.finish());
```

3. Call FPGA Kernel

```
//Write the output from the FPGA to the SSD
if (pwrite(nvmeFd, (void*)p2pPtrWrite_out_data_final, vector_size_bytes_read, BLOCK_SIZE_BYTES * OUT_FINAL_OFFSET) < 0) { //Error if <0
        printf("WRITE FAIL! %s\n", strerror(errno));
        exit(EXIT_FAILURE);
}
```

4. Write to SSD from FPGA