



Evaluation of oneAPI for FPGAs



Presented by: Nicholas Miller

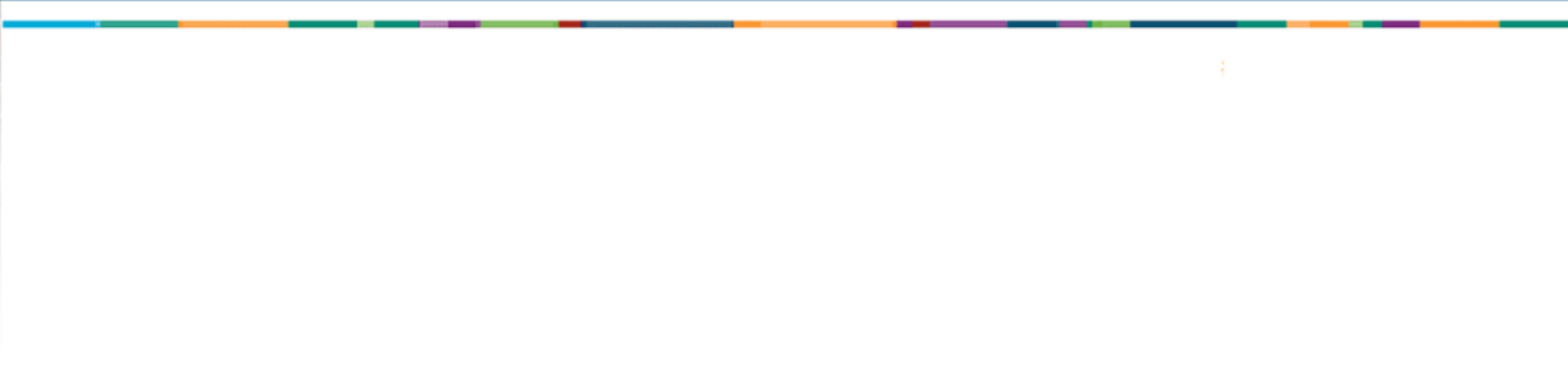
Nicholas Miller, Jeanine Cook, and Clayton Hughes



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.



Introduction



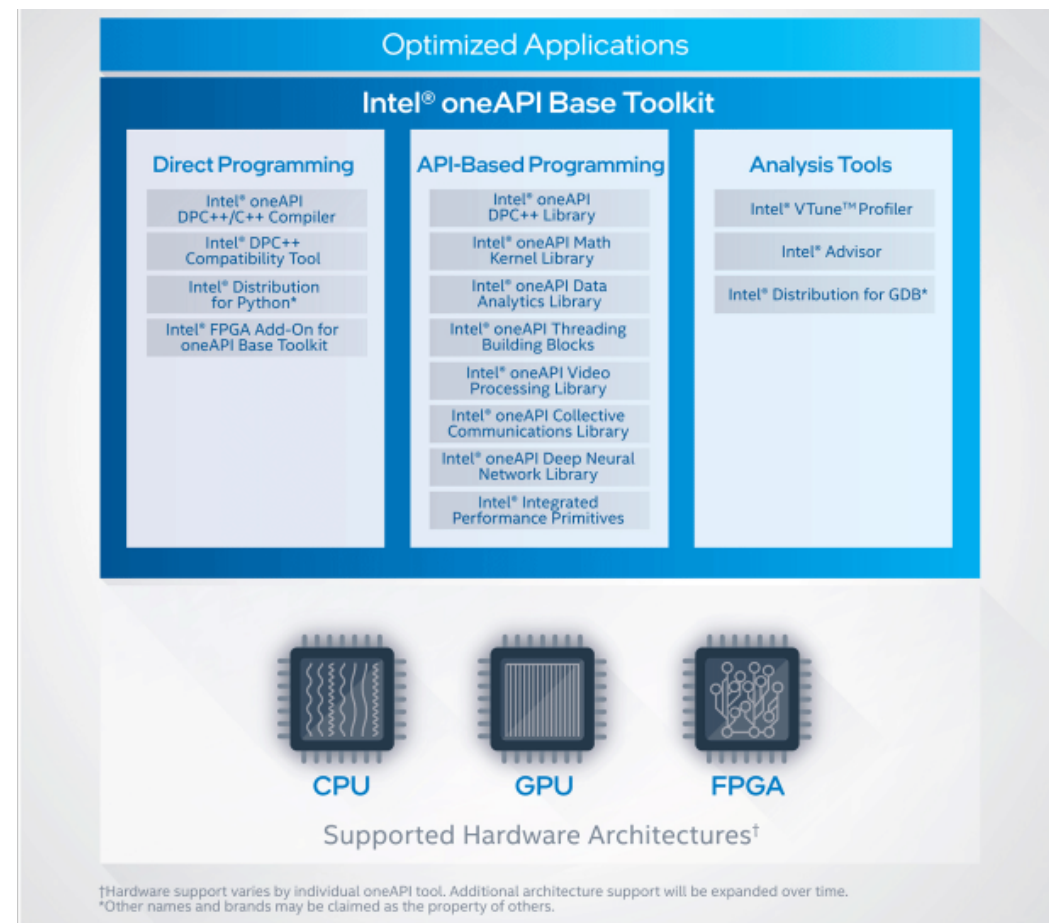


- Application-specific accelerators have shown promising results in both power and performance
 - Have historically faced challenges for HPC
 - Development environment not amenable to agile application and hardware co-design
 - System integration and deployment complexity
- Chip specialization and investment in high-level synthesis tools have brought ASA technology much closer to practical reality

*Evaluate Intel's oneAPI tools for FPGA
Programmability and Performance*

Introduction to oneAPI

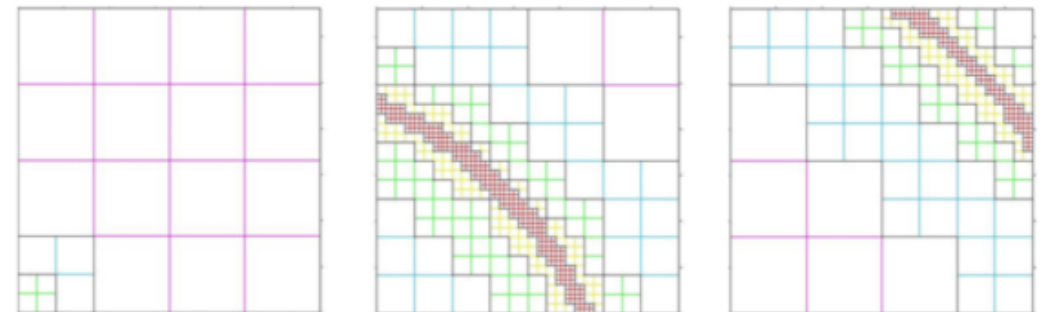
- Programming framework that provides a single interface for multiple targets
 - DPC++ which builds upon SYCL
 - Can target GPUs, CPUs, and FPGAs
- Includes libraries to accelerate certain application domains
- Open specification



<https://software.intel.com/content/www/us/en/develop/tools/oneapi/commercial-base.html#gs.3lc6t2>

- Adaptive mesh refinement proxy application
- Simulates an object moving through a mesh and adaptively refines the mesh in order to save on computation
- Computation is a simple 7-point stencil which takes an average
- Only the computation-heavy stencil calculation is moved to the FPGA
 - Mesh refinement and communication sections of the program stay the same

```
for some number of timesteps do  
  for some number of stages do  
    communicate ghost values between blocks  
    perform stencil calculation on variables  
    if stage for checksums then  
      perform checksum calculations  
      compare checksum values  
    end if  
  end for  
  if time for refinement then  
    refine mesh  
  end if  
end for
```





```

1  for (int in = 0; in < sorted_index[num_refine + 1]; in++) {
2      bp = &blocks[sorted_list[in].n];
3      for (var = 0; var < var_max; var++) {
4          sycl::range<1> num_array{ static_cast<size_t>((x_block_size + 2) *
5              (y_block_size + 2) * (z_block_size + 2)) };
6          //create a buffer that goes to the fpga
7          double* inputArray = new double[(x_block_size + 2) *
8              (y_block_size + 2) * (z_block_size + 2)];
9          //create a buffer that comes from the fpga
10         double* outputArray = new double[(x_block_size + 2) *
11             (y_block_size + 2) * (z_block_size + 2)];
12         //flatten the 4d array to a 1d array for the buffer
13         for (int i = 0; i <= x_block_size + 1; i++)
14             for (int j = 0; j <= y_block_size + 1; j++)
15                 for (int k = 0; k <= z_block_size + 1; k++)
16                     inputArray[k + (z_block_size + 2) * (j + (y_block_size + 2) * i)]
17                         = bp->array[var][i][j][k];
18         sycl::buffer<double, 1> input_buffer(inputArray, num_array);
19         {
20             sycl::buffer<double, 1> output_buffer(outputArray, num_array);
21             fpga_kernel(input_buffer, output_buffer);
22         } //output_buffer destructor called here
23         //write the data back to the block array
24         for (int i = 1; i <= x_block_size; i++)
25             for (int j = 1; j <= y_block_size; j++)
26                 for (int k = 1; k <= z_block_size; k++)
27                     bp->array[var][i][j][k] = outputArray[k + (z_block_size + 2) *
28                         (j + (y_block_size + 2) * i)];
29     } //input_buffer destructor called here
30 }

```

Create temporary arrays to hold data going to and coming from the FPGA

Buffers only accept 1D arrays so flatten the 3D array

Create the SYCL buffer

Call the FPGA kernel

Expand the returned data and store it in the host arrays



```

1 void fpga_kernel(sycl::buffer<double, 1>& input_buffer ,
2 sycl::buffer<double, 1>& output_buffer) {
3     //Device queue submit
4     queue_event = device_queue.submit([&](sycl::handler& cgh) {
5         //Create FPGA side accessors to the buffers
6         auto accessor_in =
7             input_buffer.get_access<sycl::access::mode::read_write>(cgh);
8         auto accessor_out =
9             output_buffer.get_access<sycl::access::mode::discard_write>(cgh);
10        cgh.single_task<class Stencil_kernel>(&[=]() {
11            double work[12][12][12];
12            double local_array[12][12][12];
13            for (int i = 0; i <= 11; i++)
14                for (int j = 0; j <= 11; j++)
15                    for (int k = 0; k <= 11; k++)
16                        local_array[i][j][k] = accessor_in[i][j][k];
17            for (int i = 1; i <= 10; i++)
18                for (int j = 1; j <= 10; j++)
19                    for (int k = 1; k <= 10; k++)
20                        work[i][j][k] = (
21                            local_array[i - 1][j][k] +
22                            local_array[i][j - 1][k] +
23                            local_array[i][j][k - 1] +
24                            local_array[i][j][k] +
25                            local_array[i][j][k + 1] +
26                            local_array[i][j + 1][k] +
27                            local_array[i + 1][j][k]) / 7.0;
28            for (int i = 1; i <= 10; i++)
29                for (int j = 1; j <= 10; j++)
30                    for (int k = 1; k <= 10; k++)
31                        accessor_out[i][j][k] = work[i][j][k];
32        });
33    }

```

Create accessors to get data from the host

Create local memory to store variables within the programmable fabric

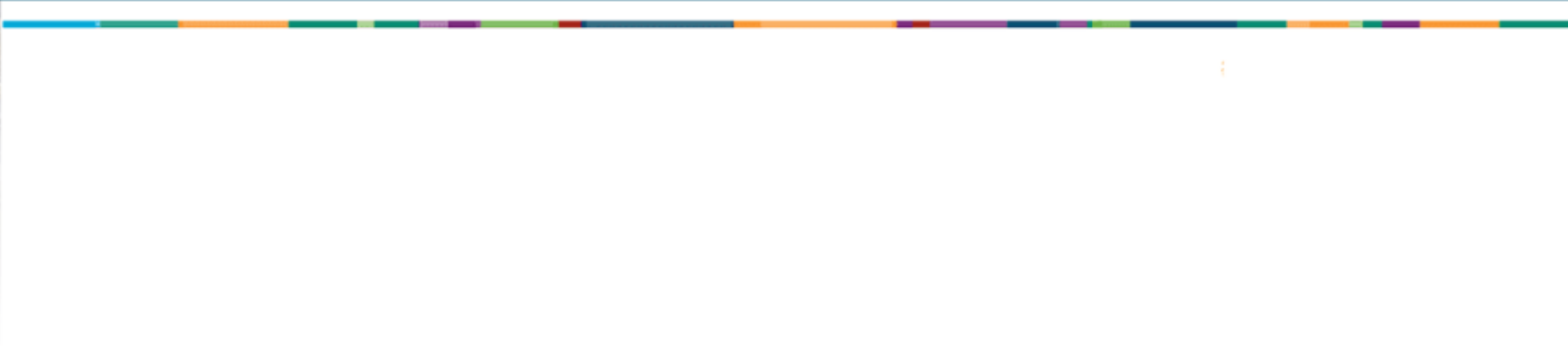
Load data from FPGA SDRAM (global memory) into programmable logic BRAM (local memory)

Compute a 7-point stencil using the local memory

Store data from local memory to global memory



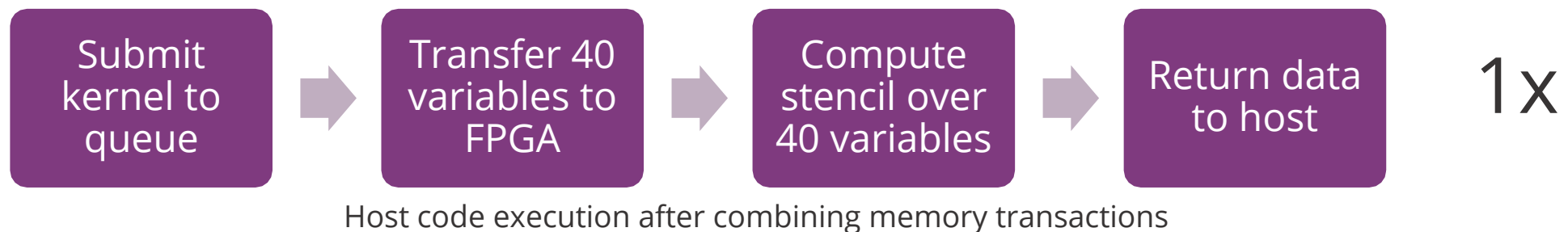
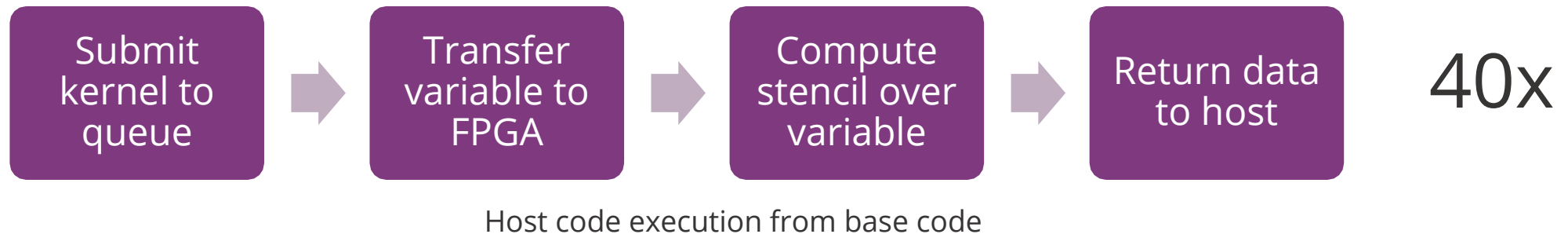
Optimizations



Combining Memory Transactions



- The optimization that provided the largest performance boost was to combine all the variable computations in a block into a single communication and computation step
- This reduced the number of calls to the SYCL runtime by 40x





- Packed *all* variables into a single array sent to the FPGA

```

1  for (int in = 0; in < sorted_index[num.refine + 1]; in++) {
2    bp = &blocks[sorted_list[in].n];
3    sycl::range<1> num_array{ static_cast<size_t>(var_max * (x_block_size + 2) *
4      (y_block_size + 2) * (z_block_size + 2)) };
5    //create a buffer that goes to the fpga
6    double* inputArray = new double[var_max * (x_block_size + 2) *
7      (y_block_size + 2) * (z_block_size + 2)];
8    //create a buffer that comes from the fpga
9    double* outputArray = new double[var_max * (x_block_size + 2) *
10     (y_block_size + 2) * (z_block_size + 2)];
11   //flatten the 4d array to a 1d array for the buffer
12   for (var = 0; var < var_max; var++)
13     for (int i = 0; i <= x_block_size + 1; i++)
14       for (int j = 0; j <= y_block_size + 1; j++)
15         for (int k = 0; k <= z_block_size + 1; k++)
16           inputArray[(var * (x_block_size + 2) * (y_block_size + 2) *
17             (z_block_size + 2)) + (k + (z_block_size + 2) *
18               (j + (y_block_size + 2) * i))] = bp->array[var][i][j][k];
19   sycl::buffer<double, 1> input_buffer(inputArray, num_array);
20   {
21     sycl::buffer<double, 1> output_buffer(outputArray, num_array);
22     fpga_kernel(input_buffer, output_buffer);
23   }
24   //write the data back to the block array
25   for (var = 0; var < var_max; var++)
26     for (int i = 1; i <= x_block_size; i++)
27       for (int j = 1; j <= y_block_size; j++)
28         for (int k = 1; k <= z_block_size; k++)
29           bp->array[var][i][j][k] = outputArray[(var * (x_block_size + 2) *
30             (y_block_size + 2) * (z_block_size + 2)) +
31             (k + (z_block_size + 2) * (j + (y_block_size + 2) * i))];
32 }

```

Kernel Changes - Combining Memory Transactions

- Compute on all 40 variables in each kernel call
 - First bring all 40 variables into local memory
 - Then compute the stencil of on all 40 variables

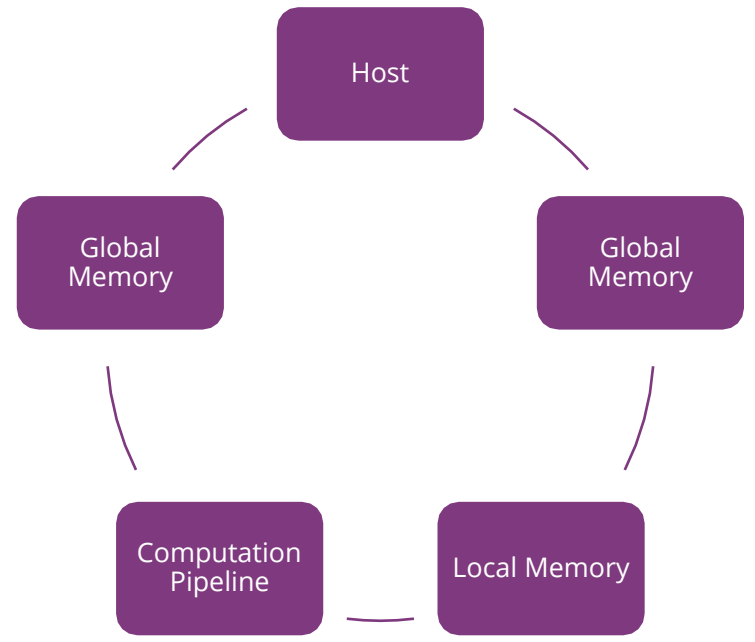
```

1 void fpga_kernel(sycl::buffer<double, 1>& input_buffer ,
2 sycl::buffer<double, 1>& output_buffer) {
3     //Device queue submit
4     queue_event = device.queue.submit([&](sycl::handler& cgh) {
5         //Create FPGA side accessors to the buffers
6         auto accessor_in =
7             input_buffer.get_access<sycl::access::mode::read_write>(cgh);
8         auto accessor_out =
9             output_buffer.get_access<sycl::access::mode::discard_write>(cgh);
10        cgh.single_task<class Stencil_kernel>([=]() {
11            //create a local copy of the array data for increased performance
12            double local_array[40][12][12][12];
13            for (int var = 0; var < 40; var++)
14                for (int i = 0; i <= 11; i++)
15                    for (int j = 0; j <= 11; j++)
16                        for (int k = 0; k <= 11; k++)
17                            local_array[var][i][j][k] =
18                                accessor_in[(var * (12) * (12) * (12)) + (k + (12) *
19                                    (j + (12) * i))];
20            for (int var = 0; var < 40; var++)
21                for (int i = 1; i <= 10; i++)
22                    for (int j = 1; j <= 10; j++)
23                        for (int k = 1; k <= 10; k++)
24                            accessor_out[(var * (12) * (12) * (12)) + (k + (12) *
25                                (j + (12) * i))] = (
26                                local_array[var][i - 1][j][k] +
27                                local_array[var][i][j - 1][k] +
28                                local_array[var][i][j][k - 1] +
29                                local_array[var][i][j][k] +
30                                local_array[var][i][j][k + 1] +
31                                local_array[var][i][j + 1][k] +
32                                local_array[var][i + 1][j][k]) / 7.0;
33        });
34    });
35 }

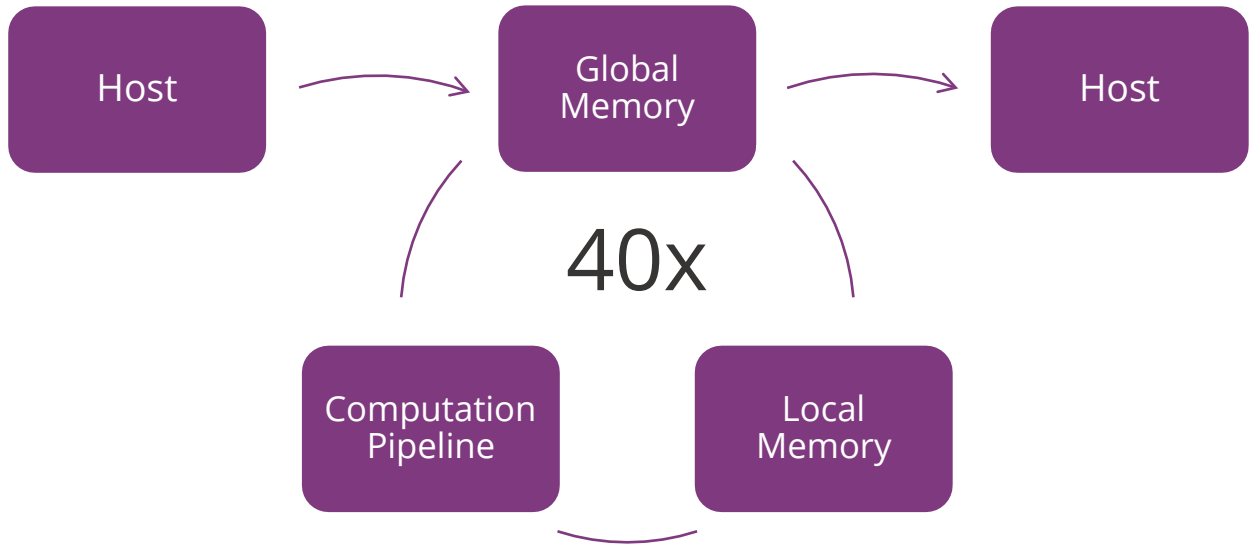
```

Reduce Local Memory Usage

- Instead of always storing all 40 variables in the kernel, only store a single variable which is being computed upon
- Reduces the overhead of the local memory usage by 40x without noticeable slowdown



Data flow in combined memory transactions code



Data flow to reduce local memory usage

FPGA Changes - Reduce Local Memory

- Compute the stencil directly after bringing a single variable into local memory
- Removed the outer loop previously at line 20

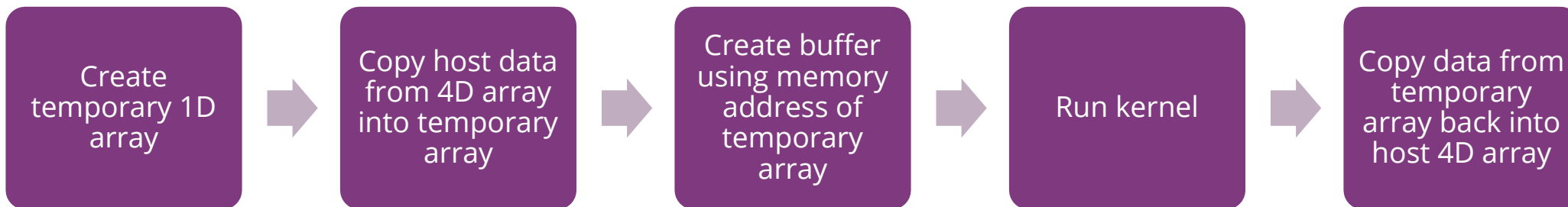
```

1 void fpga_kernel(sycl::buffer<double, 1>& input_buffer ,
2 sycl::buffer<double, 1>& output_buffer) {
3     //Device queue submit
4     queue_event = device.queue.submit([&](sycl::handler& cgh) {
5         //Create FPGA side accessors to the buffers
6         auto accessor_in =
7             input_buffer.get_access<sycl::access::mode::read_write>(cgh);
8         auto accessor_out =
9             output_buffer.get_access<sycl::access::mode::discard_write>(cgh);
10        cgh.single_task<class Stencil_kernel>([=]() {
11            //create a local copy of the array data for increased performance
12            double local_array[12][12][12];
13            for (int var = 0; var < 40; var++)
14                for (int i = 0; i <= 11; i++)
15                    for (int j = 0; j <= 11; j++)
16                        for (int k = 0; k <= 11; k++)
17                            local_array[i][j][k] =
18                                accessor_in[(var * (12) * (12) * (12)) + (k + (12) *
19                                    (j + (12) * i))];
20            for (int i = 1; i <= 10; i++)
21                for (int j = 1; j <= 10; j++)
22                    for (int k = 1; k <= 10; k++)
23                        accessor_out[(var * (12) * (12) * (12)) + (k + (12) *
24                            (j + (12) * i))] = (
25                            local_array[i - 1][j][k] +
26                            local_array[i][j - 1][k] +
27                            local_array[i][j][k - 1] +
28                            local_array[i][j][k] +
29                            local_array[i][j][k + 1] +
30                            local_array[i][j + 1][k] +
31                            local_array[i + 1][j][k]) / 7.0;
32        });
33    });
34 }

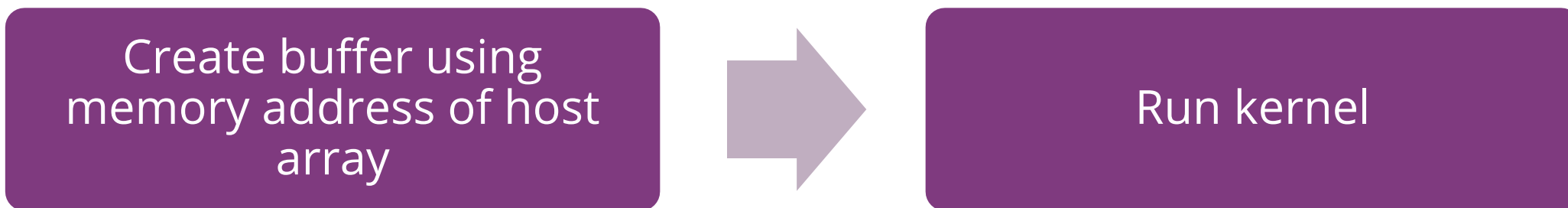
```

Flattening Arrays

- Converted all multi-dimensional arrays in the host code to 1d, which eliminated the need to convert for the buffer creation
- Reduces host side pre- and post- processing needed on every kernel invocation



Host code execution from combined memory transactions code



Host code execution after flattening arrays

Host Changes - Flattening Arrays

- Completely removes the need for packing the host data into arrays
- Uses the original host side arrays memory locations for buffer creation so the data is copied from and to the FPGA more efficiently
- Now only the following steps are needed:
 1. Create the buffer using the host memory address
 2. Enqueue the kernel to be run on the FPGA
- No modifications required in kernel code

```

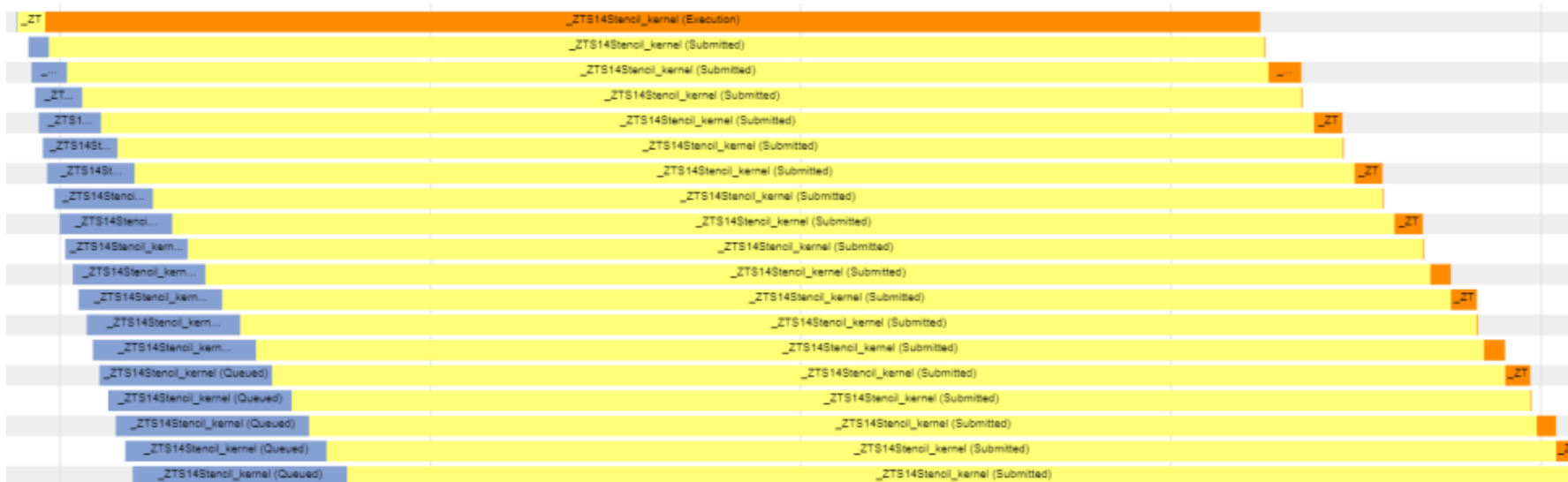
1  for (int in = 0; in < sorted_index[num_refine + 1]; in++) {
2      bp = &blocks[sorted_list[in].n];
3      sycl::range<1> num_array{ static_cast<size_t>(var_max *
4          (x_block_size + 2) * (y_block_size + 2) * (z_block_size + 2)) };
5      {
6          sycl::buffer<double, 1> input_buffer(bp->array, num_array);
7          fpga_kernel(input_buffer);
8      }
9  }

```

Buffering SYCL Runtime Calls



- Calling the SYCL runtime for the kernel queues and submits it while the FPGA works on the stencil calculation
- This only works if the execution of the kernel is long enough to hide the SYCL runtime overheads



- Blue: Queued for submission
- Yellow: Submitted and waiting to run
- Orange: Executing

Host Changes - Buffering SYCL Runtime Calls



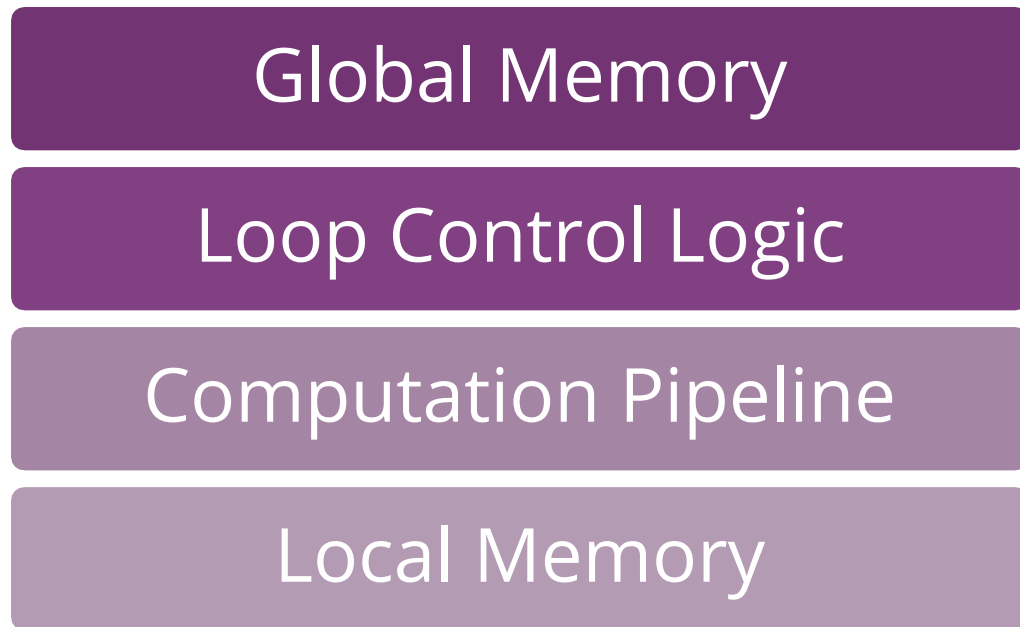
- Create a vector that stores the buffers of each kernel invocation
- All buffer destructors are called when the function exits, and the vector destructor is called
- Data is transferred back as needed – controlled by the SYCL runtime until destructor is called
- If only using a single block there is no functional change as we are queuing kernel invocations over blocks

```
1  std::vector<sycl::buffer<double, 1>> input_buffer;  
2  
3  for (int in = 0; in < sorted_index[num_refine + 1]; in++) {  
4      bp = &blocks[sorted_list[in].n];  
5      input_buffer.push_back(sycl::buffer<double, 1>(bp->array,  
6          sycl::range<1>(static_cast<size_t>(var_max * (x_block_size + 2) *  
7          (y_block_size + 2) * (z_block_size + 2))));  
8      fpga_kernel(input_buffer[in]);  
9  }
```

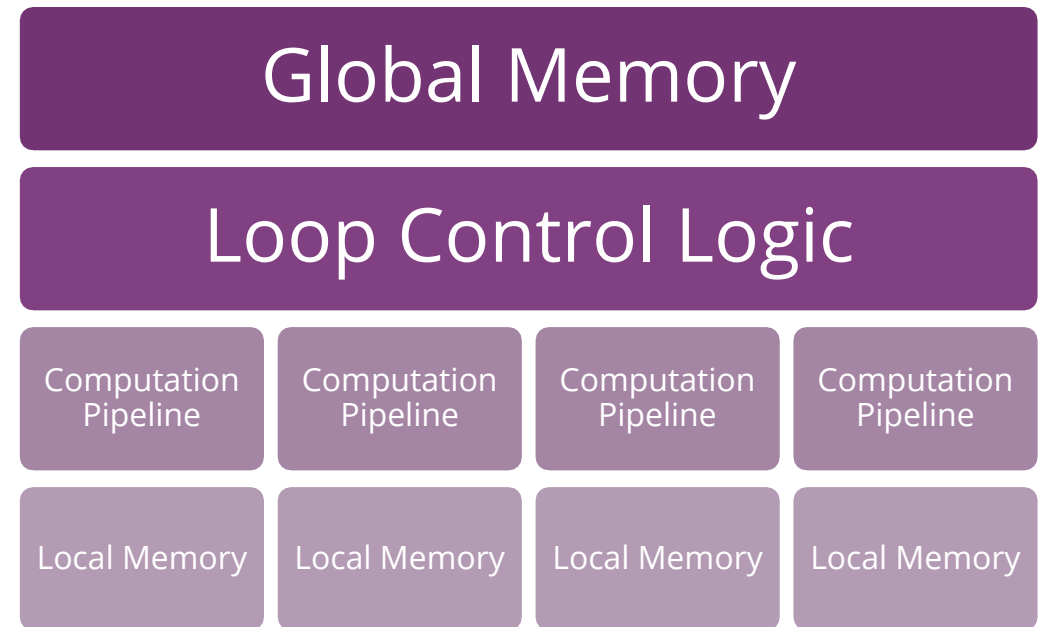
Unrolling Computation Pipeline Loop



- Unroll the outermost loop to create multiple concurrent pipelines
 - Each variable computed independently



Simplified FPGA architecture of reduced local memory optimization code



Simplified FPGA architecture of unrolled outermost computation loop

FPGA Changes - Unrolling Computation Pipeline Loop

- Compiler hint to inform unroll depth

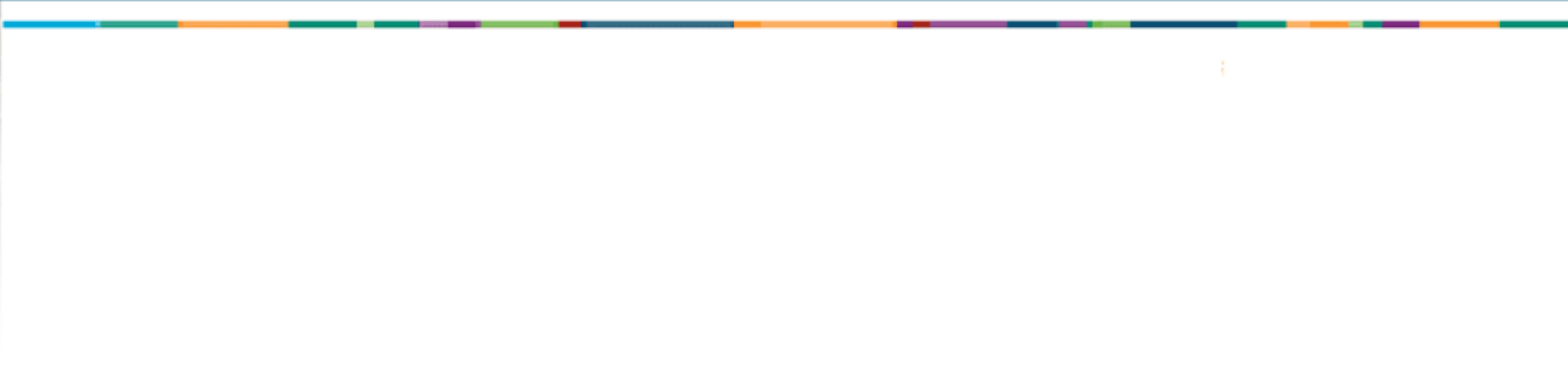
```

1 void fpga_kernel(sycl::buffer<double, 1>& input_buffer) {
2     //Device queue submit
3     queue_event[kernelCounter % 2] = device_queue.submit([&](sycl::handler& cgh)
4     {
5         //Create accessors
6         auto accessor_in =
7             input_buffer.get_access<sycl::access::mode::read_write>(cgh);
8         cgh.single_task<class Stencil_kernel>([=]() {
9             double local_array[12][12][12];
10            #pragma unroll X //replace X with the number of unrolls 0, 2, 4, or 8
11            for (int var = 0; var < 40; var++) {
12                for (int i = 0; i <= 11; i++)
13                    for (int j = 0; j <= 11; j++)
14                        for (int k = 0; k <= 11; k++)
15                            local_array[i][j][k] = accessor_in[(var * (12) * (12) *
16                                (12)) + (k + (12) * (j + (12) * i))];
17                for (int i = 1; i <= 10; i++)
18                    for (int j = 1; j <= 10; j++)
19                        for (int k = 1; k <= 10; k++)
20                            accessor_in[(var * (12) * (12) * (12)) + (k + (12) *
21                                (j + (12) * i))] = (
22                                local_array[i - 1][j][k] +
23                                local_array[i][j - 1][k] +
24                                local_array[i][j][k - 1] +
25                                local_array[i][j][k] +
26                                local_array[i][j][k + 1] +
27                                local_array[i][j + 1][k] +
28                                local_array[i + 1][j][k]) / 7.0;
29                    }
30                });
31            });
32 }

```



Results



Experimental Setup

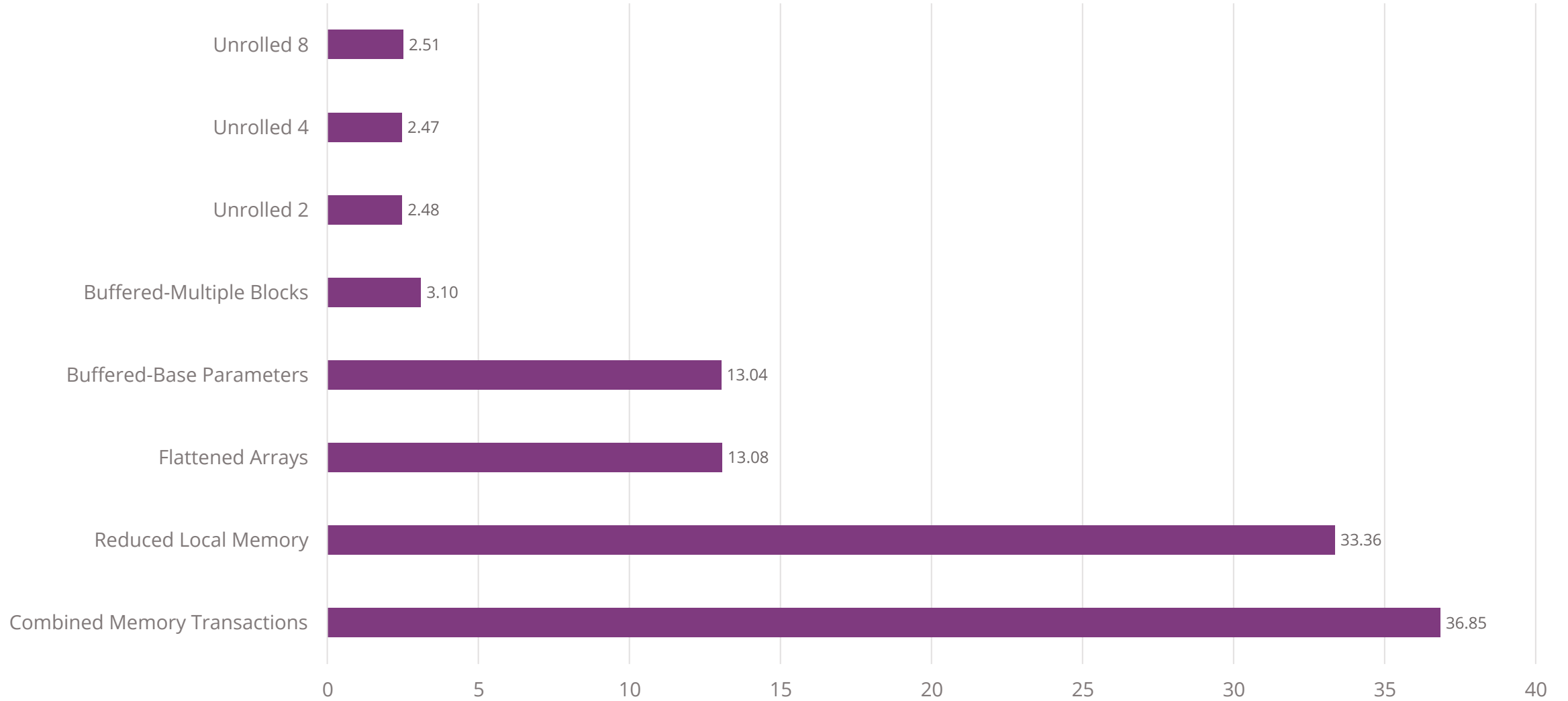


- Run on the Intel Devcloud system
- Submitted to node by OpenPBS scheduler
- Increased number of blocks in a run to compare for the buffering tests

CPU	2 x Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz
FPGA Family	Arria 10
FPGA Device	10AX115S2F45I2SGES
oneAPI Version	Beta08
System Memory	196 GB
Base Parameters	No Parameters
Increased Blocks Parameters	<code>--num_refine 4 --max_blocks 9000 --num_objects 1 --object 2 0 -1.71 -1.71 -1.71 0.04 0.04 0.04 1.7 1.7 1.7 0.0 0.0 0.0 --num_tsteps 25</code>

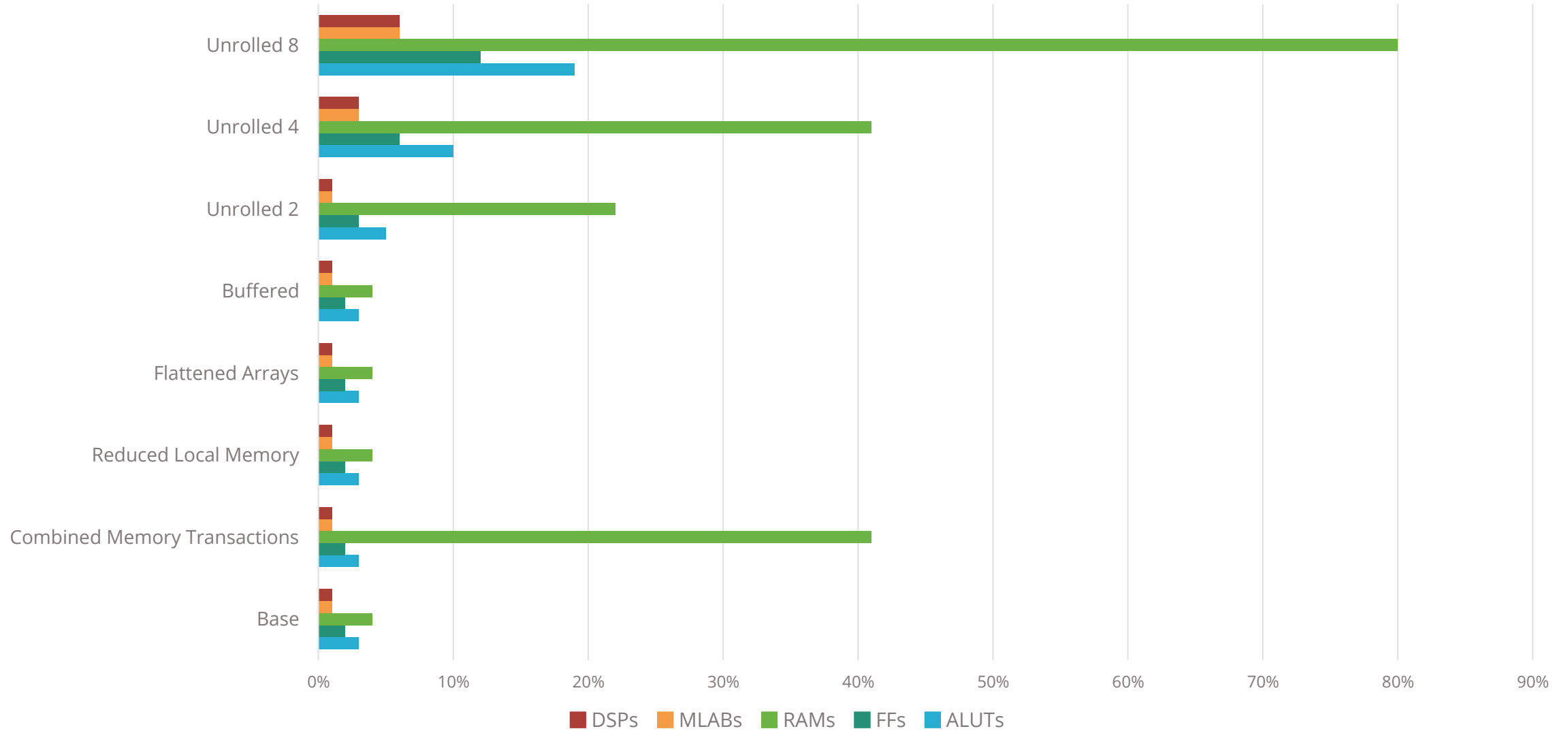


Slowdown Compared to Processor





Device Resource Usage



Summary



Manufacturing and materials advances have brought application-specific accelerators closer to reality

Evaluated the miniAMR proxy application using Intel's oneAPI tools to determine maturity and viability of HLS for ASA development

Showed that application was easy to port but difficult to optimize





Questions?

