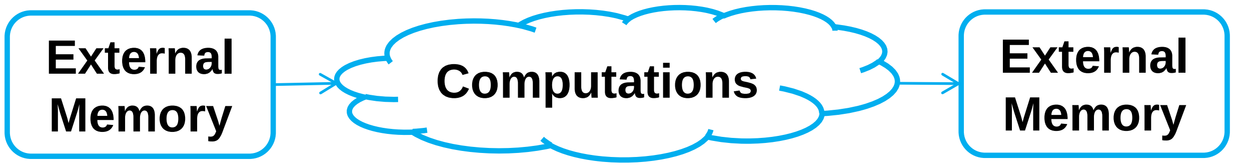
Being careful with memory accesses, Part 1

Dmitry Denisenko

Being careful with memory accesses

Altera OpenCL platforms are built to provide very high external memory bandwidth. However, no amount of hardware engineering can compensate for careless coding. When writing OpenCL kernels for FPGAs, be mindful of every external memory access. An ideal kernel:

- reads its inputs from external memory once, possibly into local memory or shift registers,
- processes it, and
- writes output once to external memory.



Unnecessarily going to external memory in the middle of computations, or reading input data multiple times, is a sure way to limit your kernel's performance.

If you are using I/O channels to bring data into the kernel instead of external memory, great job! Chances are, your I/O interfaces are a lot less forgiving than external memory and you already structured your computation efficiently.

Being careful with memory accesses

How to tell if you have a memory bandwidth problem?

One way is to compile your kernel to hardware with profiling enabled, and then use AOCL Profiler GUI to see if any of your memory accesses are inefficient (Never heard of AOCL Profiler? Check out our Best Practices Guide!).

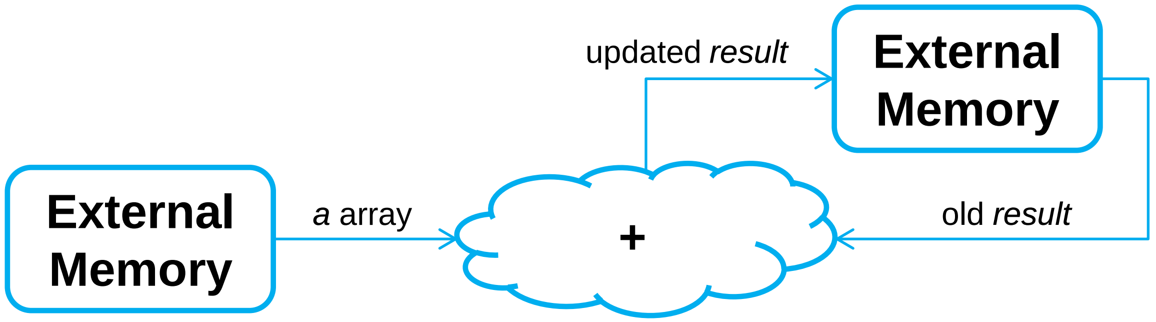
However, you don’t need to go through the full hardware compile to know if your kernel is structured well. Just find all the places where you dereference your global pointers (every `[]` and `*ptr` calls). Drawing a data flow diagram can be helpful in visualizing the code structure.

Being careful with memory accesses

For example, you may write a simple summation kernel as follows:

```
kernel void do_sum (global int *a, int n, global int *result) {
  for (int i=0; i<n; i++) {
    *result += a[i];
  }
}
```

If you draw data flow diagram for this kernel, it may look something like this:



This clearly looks inefficient. The `result` value needs to make full round-trip to the external memory before it can be updated once!

☞ In case you did not carefully think about your memory accesses for this kernel, Optimization Report will tell you about the inefficiency and its cause. However, Optimization Report will not always be so nice!

ALTERA® MEASURABLE ADVANTAGE™

Being careful with memory accesses

The example above is easy to fix – just have a local variable (a register!) store intermediate sum value and only write it to global `result` variable once at the end.

No matter what hardware architecture or programming language you use, memory bandwidth will often be the bottleneck in your kernel's performance. In following notes, we'll cover more complicated cases of memory accesses where you need to use local memories, shift registers, caches, and other fun memory concepts!