MP-STREAM: A Memory Performance Benchmark for Design Space Exploration on Heterogeneous HPC Devices

***Abstract*—Sustained memory throughput is a key determinant of performance in HPC devices. Having an accurate estimate of this parameter is essential for manual or automated design space exploration for any HPC device. While there are benchmarks for measuring the sustained memory bandwidth for CPUs and GPUs, such a benchmark for FPGAs has been missing. We present MP-STREAM, an OpenCL-based synthetic micro-benchmark for measuring sustained memory bandwidth, optimized for FPGAs, but which can be used on multiple platforms. Our main contribution is the introduction of various generic as well as device-specific parameters that can be tuned to measure their effect on memory bandwidth. We present results of running our benchmark on a CPU, a GPU and two FPGA targets, and discuss our observations. The experiments underline the utility of our benchmark for optimizing HPC applications for FPGAs, and provide valuable optimization hints for FPGA programmers.**

I. Introduction

High-Performance Computing (HPC) machines are becoming increasingly heterogeneous with the adoption of GPUs, many-core accelerators like Xeon Phis, and more recently, FPGAs. For FPGAs to become truly mainstream, an ecosystem of tools and benchmarks will be required. One such requirement is to have a benchmark that can measure the memory bandwidth achievable with FPGA target devices and associated tools. Memory bandwidth is increasingly the performance bottleneck for many HPC applications. Out of thirteen *dwarfs* of HPC identified in [1], seven can be considered memory-bound. Benchmarks that can qualify HPC devices with respect to sustained memory bandwidth can be of immense utility.

The STREAM benchmark is the de-facto industry standard for measurement of computer memory bandwidth, and has been widely used and cited since its introduction in 1995 [2]. Its implementation for GPUs using CUDA and OpenCL has also been developed as GPU-STREAM [3]. To the best of our knowledge, the STREAM benchmark has not yet been adapted for evaluating FPGA devices. Our work aims primarily to address this.

In this paper, we present our interpretation of the STREAM benchmark, which we call Multi-Platform-STREAM, or simply MP-STREAM. Our purpose is to enable exploring the entire design-space that effects the sustained memory-bandwidth. Our key contribution the introduction of tuning parameters that have an impact on the sustained memory bandwidth. These additional features provide valuable insights to optimizing high-performance code for FPGAs. The MP-STREAM benchmark is not limited to FPGAs though and we present results across four different architecture in this paper. We discuss some interesting and – at times – non-intuitive results that appear when the tuning parameters are explored, emphasizing the observation that memory-bandwidth optimization is both target device and vendor specific, and that cross-platform parallel programming languages like OpenCL are not always performance-portable.

II. Background and Requirement of a New Benchmark

STREAM [2] is a synthetic benchmark, originally written in Fortran 77, for measuring the performance of four different kernels performing array operations. These four kernels are:

1) COPY: `a(i) = b(i)`
2) SCALE: `a(i) = q*b(i)`
3) SUM: `a(i) = b(i) + c(i)`
4) TRIAD: `a(i) = b(i) + q*c(i)`

where `q` is a scalar. By knowing the size of the arrays, the size of each element, and then measuring the time taken to execute a kernel, the sustained memory-bandwidth is computed. The STREAM benchmark has become the de-facto industry standard for reporting the sustained memory bandwidth. GPU-Stream benchmark [3] implements the four kernels listed above in OpenCL and CUDA frameworks for GPGPUs. This open-source OpenCL benchmark was a useful resource in developing our FPGA-oriented version.

The original STREAM benchmark was written for CPUs and cannot be directly ported to FPGAs. The GPU-STREAM's OpenCL benchmark could be more easily adapted for OpenCL-compatible FPGA targets, but could not be effectively used for FPGAs directly, as our purpose is to be able to fully explore the design-space of FPGA memory architecture. The sustained memory bandwidth on FPGA targets is affected by a large number of parameters – not all relevant to CPUs or GPUs – and some parameters affect the memory bandwidth in unexpected ways as we will see later.

III. Exploring the FPGA Memory-Access Architecture with the MP-STREAM Benchmark

We have developed a portable benchmark based on the four kernels defined in the original STREAM benchmark. The benchmark has been written in OpenCL [4], which is a heterogeneous parallel programming language and framework. We have tested the benchmark on four heterogeneous targets. The bandwidth is simply calculated by measuring the time taken to run each of the kernels for a known array size.

The key motivation behind extending STREAM benchmark was to introduce parameters that had an effect on sustained

memory bandwidth especially in the context of FPGAs. In this section we discuss these parameters, categorized as generic or specific to a target. The generic parameters are as follows:

Size of array: The size of the array is the only controllable parameter in the original STREAM benchmark. Smaller array sizes have a proportionally larger memory latency impact, so larger arrays measure the asymptotic performance. Also arrays should be large enough to ensure that we are in fact measuring DRAM bandwidth in cache-based targets.

Source/destination of streams: While typically bandwidth to global memory (device-DRAM) is of primary interest for performance, we have also included the ability to measure bandwidth between the host and the device which in the typical case would give us the bandwidth over a PCIe host–device interface.

Data type: The benchmark currently supports *integer* and *double* types. Using doubles for the *copy* kernel translates into a 64-bit coalesced access, while for other kernels, there is impact on the computation as well.

Degree of vectorization: OpenCL allows vector data types as arguments to the kernel which translates to a memory controller on the FPGA that coalesces memory accesses – up to 16 words – leading to improved memory throughput.

Data access pattern: Streaming from memory is equivalent to an iteration over the array, which may be multi-dimensional. The pattern of index-access has considerable impact on bandwidth. Testing different access patterns was considered a future direction for the original STREAM benchmark. The effect of access pattern on bandwidth has been investigated in a number of studies [5], [6]. In MP-STREAM we currently test two patterns: contiguous data, and strided data, with fixed stride.

Kernel loop management: The kernels of the stream benchmark are array operations, which would be implemented as loops in software. When such a kernel is ported to OpenCL, the array operation can be expressed in different ways, which translates to a different memory-access architecture and has an impact on the sustained bandwidth, at times in unexpected ways. The three types of "loop management" that can be experimented with our benchmark are as follows:

NDRange Kernel: In OpenCL, loop over a kernel's iteration-space is typically subsumed by launching multiple *work-items*, the total number of which is known as *NDRange*. An *NDRange kernel* looks like this:

```
/** HOST **/
clEnqueueNDRangeKernel (kernel, ARRAY_SIZE, ...);
/** DEVICE **/
index    = get_global_id(0);
c[index] = a[index];
```

Single work-item, flat-looping kernel: We can launch just one work-item, and have a *for* loop in the kernel, e.g.:

```
/** HOST **/
clEnqueueNDRangeKernel (kernel, 1, ...);
/** DEVICE **/
for (index=0; index < ARRAY_SIZE; index++)
  c[index] = a[index];
```

Single work-item, nested-looping kernel: We found, somewhat unexpectedly, that there is another variant in this scenario, where we loop over a 2D array in a *nested* fashion (shown below), and that *can* affect the bandwidth:

```
/** HOST **/
clEnqueueNDRangeKernel (kernel, 1, ...);
/** DEVICE **/
for (i=0; i < ARRAY_SIZE_I ; i++)
  for (j=0; j < ARRAY_SIZE_J ; j++)
    c[i][j] = a[i][j];
```

Loop unroll factor: The loop unroll factor can be controlled in OpenCL through the `__attribute__((opencl_unroll_hint(n)))` command. Our benchmark's build scripts generate custom kernel code inserting this optimizations as specified by command-line flags.

Required work-group size: The optional attribute `reqd_work_group_size(X, Y, Z)` allows the compiler to optimize the generated code, and is recommended by some OpenCL-FPGA compilers to optimize the FPGA synthesis.

FPGA device vendors often have optimization techniques that are not necessarily part of the OpenCL standard. For example, Intel's AOCL compiler has a number of optimization parameters discussed in [7]. The optimization parameters we considered relevant to this benchmark were: (1) Number of SIMD work-items and (2) Number of compute-units.

Similarly, Xilinx's SDACCEL also has custom optimizations that can be added to the kernel code for different types of optimizations, as described in [8]. The ones relevant to this benchmark are: (1) Pipeline loop, (2) Pipeline work-items, (3) Maximum memory ports, and (4) Memory port data width

IV. Experiments in Design-Space Exploration using MP-STREAM

Our experimental setup comprises two OpenCL-compatible FPGA targets, an Intel CPU and an Nvidia GPU, demonstrating the portability of our benchmark across heterogeneous targets. Some more details are as follows:

CPU: Intel Xeon CPU E5-2609 v2, 10 MB cache, 34 GB/s Peak BW.

GPU: GeForce GTX Titan Black, 336 GB/s Peak BW.

FPGA-AOCL: Altera Stratix V GS D5 (Nallatech PCIe-385), 25 GB/s Peak BW, AOCL 15.1 compiler.

FPGA-SDAccel: Xilinx Virtex 7 XC7 (Alpha-Data ADM-PCIE-V7), 10 GB/s Peak BW, Sdaccel 2015.1 compiler.

The objective behind these experiments is to demonstrate the portability of our benchmark across heterogeneous platforms, and also to emphasize the utility of the tuning parameters that we have introduced in exploring the design-space of memory-access architecture, specifically for FPGAs. We would like to emphasize that there is no guarantee we have achieved the best possible results on the tested targets, and we do not claim exhaustive exploration of the search space.

Varying Array Size and Vectorization: The first two experiments were on varying the size of the input array, as well as the size of vectors used in the kernel signature (Figure 1). The horizontal dotted-lines indicate the peak bandwidths for that target. For all targets, larger array sizes leads to better sustained bandwidth, which is an expected result, as large array sizes hide the latencies of memory access and of the control transfer between host and device. We see the bandwidths plateau around 4MB, and this is the fixed size we use when experimenting with variation across other parameters. Note that GPU has both a much higher peak bandwidth, and

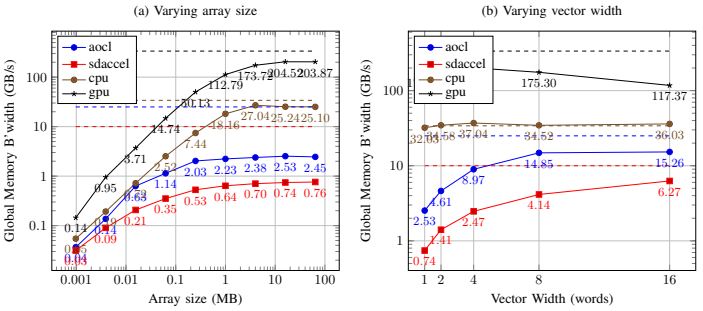

Fig. 1: Testing all four targets with varying (a) array sizes and (b) vector size (memory coalescing) for the *copy* kernel. Word size is 32 bits, and data is contiguous, loop-management is optimal for each target. No other optimizations are used.

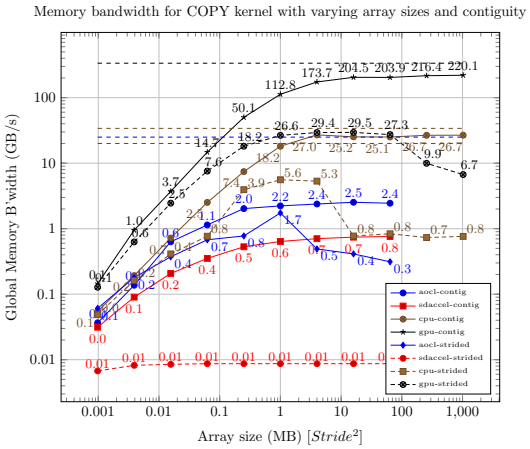

Fig. 2: Evaluating effect of data contiguity for different array sizes (and strides). Word size is 32 bits, loop-management is optimal for each target, and no vectorization or other optimizations are used.

a sustained bandwidth very close to the peak. The FPGA targets' sustained performances are relatively worse compared to their peaks. If we look at Figure 1(b) though where we use vectors to coalesce memory accesses, we see the FPGA targets approaching their peak bandwidths, and it is obvious that unless we use coalesced memory access, we are severely under-utilizing the available memory bandwidth *on FPGAs*.

Effect of Data Contiguity: The effect of data contiguity on sustained memory bandwidth to a DRAM is well known. We tested two patterns of access: contiguous, and strided with a fixed stride. For strided access, we accessed the row-major 2D array in a column-major fashion. The results are shown in Figure 2. As expected, all targets see a deterioration in performance for strided access, although to varying degrees. One can see potential for a possible optimization where – if there are multiple strided accesses to the same array(s) in global memory as is common in scientific applications – it may be worthwhile re-arranging data at the host to convert subsequent strided accesses to contiguous accesses.

Different Loop Managements: As discussed earlier, we tested three different ways of expressing array access loop,

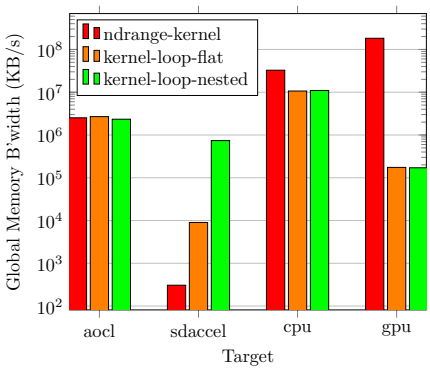

Fig. 3: Evaluating the effect of different kinds of loop management on all four targets. Array size is 4MB, word size is 32 bits, and no vectorization or other optimizations are used.

and results are shown in Figure 3. The GPU and CPU perform best for the *NDRange Kernel*, where multiple *work-items* are launched. This makes best use of the data parallelism available in these targets, and a typical OpenCL programmer can be expected to write applications in this manner. Consistent better performance on FPGA targets however is better achieved by having a single-work-item kernel, with a *for* loop at the kernel. The SDACCEL target is quite sensitive to this parameter, and surprisingly shows much better performance when we access the 2D array in a nested looping fashion, which implies that the memory-access logic is synthesized differently, even if the eventual underlying access pattern is exactly the same. One can reasonably presume that experimenting with more devices will show similar target-specific characteristics, which supports our argument that a synthetic benchmark for exploring the design-space of the memory-access architecture is critical for achieving performance on HPC systems.

Testing All Four Kernels: The STREAM benchmark has four kernels as discussed earlier: *copy, scale, add* and *triad*. Though our focus is on the *copy* kernel for evaluating the bandwidth, we did run all four kernels on all four targets to qualify our benchmark. The results are shown in Figure 4(a). Since all four kernels are quite simple, the performance could be expected to be memory-bound, and in general we can see this effect in the results.

Device Specific Optimizations: While we have not exhausted all possible combination of optimizations on the FPGA targets, we do consider them when defining our designs-space, and our build framework allows the user to specify target-specific flags. We ran some experiments on one target (AOCL), to see how well they compare when we try to achieve the same effect using native OpenCL features of using vectorized data types. As we can see from the results in Figure 4(b), the native vectorization optimization leads to more reliable improvement in performance. The other two optimizations, increasing the number of SIMD items, and increasing the number of compute-units, which express parallelism as well, have less consistent results, eventually giving poorer performance as we increase their scale. We also found that the AOCL optimizations take up more FPGA resources when compared with equivalent native OpenCL optimizations. We feel this

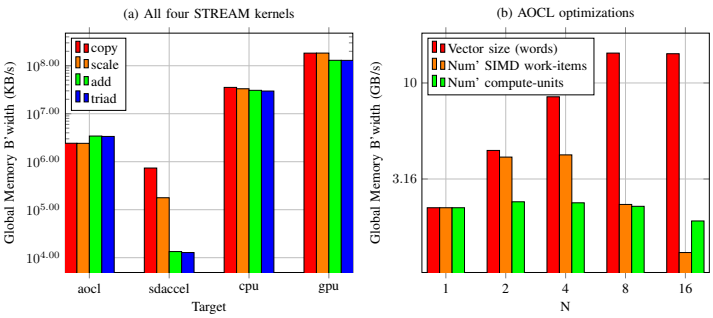

Fig. 4: (a) Testing all four STREAM kernels on all targets. (b) Evaluating two Altera-OCL specific optimizations against a generic vectorization optimization. In (b), *N* represents either the vector width, the number of SIMD work-items, or the number of compute-units (see legend). For both, array size is 4MB, and word size is 32 bits. Data is accessed contiguously.

is an argument in favour of exploring memory-access design space using native OpenCL whenever possible. We do not have a good explanation for why these opaque vendor specific optimizations generally perform worse than native OpenCL.

General Observations

An obvious – though perhaps unsurprising – observation is that OpenCL is not always performance portable across heterogeneous devices, even if it be source-code portable. In fact, even if we look at FPGA-only targets, Intel/Altera and Xilinx frameworks show different behaviours for the same set of parameters. Target-specific domain expertise is thus needed for getting the best out of each architecture. Smarter optimizing compilers would be very useful too to make performance on these heterogeneous targets easier to achieve. Both a manual and automated design-space exploration route will benefit from a benchmark that fully explores the memory-access design-space, leading to our work on MP-stream.

Looking at the comparative picture across four targets, it is clear that GPUs remain far ahead of the curve in both peak and sustained memory bandwidth, and will out-perform in memory-bound applications. FPGA vendors seeking to compete with GPUs in the memory-bound HPC area have considerable catching-up to do before they can be considered viable alternatives. FPGAs are moving towards mainstream HPC and have been shown to have better performance in a select set of scientific computing problems. What we have not considered in this paper is the energy-efficiency of the devices, but that is one area where FPGAs can still win in spite of the higher achievable bandwidths on GPUs. Also, the introduction of high-throughput Hybrid-Memory Cube on FPGA boards which have much higher peak bandwidths can change the picture we present in this paper considerably. FPGA-OpenCL tools can also be expected to mature over time and show more consistent memory performance that takes into account different coding styles.

Another useful takeaway is that if data is accessed repeatedly across many iterations, as is common scientific applications e.g. in case of a time loop over space in a weather model, then there is strong case to be made for pre-shaping that data to a format that leads to most efficient access from the acceleration device. Finally, we consider this a very useful insight that optimizations involving native OpenCL seem to perform better than propriety, device-specific optimizations.

V. Conclusion

We have presented a new benchmark called MP-STREAM, which extends the original STREAM benchmark for FPGA targets. Our work is driven by our effort to build an optimizing compiler for FPGAs, but this benchmark is a stand-alone project which we hope would play a useful role in FPGA transition to mainstream computing.

We made the case that FPGAs required an extension of the original STREAM and the GPU-STREAM benchmarks as there are a number of additional tuning parameters that effect FPGA memory bandwidth. We have presented this as the key contribution of our work, and our results clearly show that there is indeed significant impact of such parameters. In the course of our experiments, the observations we made about achieving high memory bandwidth on heterogeneous targets were discussed as well which can lead to better performing HPC applications on FPGAs. In the future, we plan to update our results with newer FPGA boards and OpenCL compiler versions, and also introduce more optimizations across all targets. We have made this benchmark publicly available[1], and plan to have a website for users to contribute and share their results with the wider community.

References

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

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

[3] T. Deakin and S. McIntosh-Smith, "Gpu-stream: Benchmarking the achievable memory bandwidth of graphics processing units," in *Poster session presented at IEEE/ACM SuperComputing,*, Austin, TX, USA, 2015.

[4] "Opencl: The open standard for parallel programming of heterogeneous systems," https://www.khronos.org/opencl/, accessed: 2016-06-15.

[5] B. Jang, D. Schaa, P. Mistry, and D. Kaeli, "Exploiting memory access patterns to improve memory performance in data-parallel architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 105–118, Jan 2011.

[6] S. Byna, X.-H. Sun, W. Gropp, and R. Thakur, "Predicting memory-access cost based on data-access patterns," in *Cluster Computing, 2004 IEEE International Conference on*, Sept 2004, pp. 327–336.

[7] Altera, "Altera sdk for opencl, best practices guide," Altera, Tech. Rep., Apr 2015, last accessed on 15th June 2016. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/opencl-sdk/aocl_optimization_guide.pdf

[8] Xilinx, "Sdaccel development environment, user guide," Xilinx, Tech. Rep., Oct 2015, last accessed on 15th June 2016. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_3/ug1023-sdaccel-user-guide.pdf

[1] https://github.com/waqarnabi/mp-stream