Design Space Exploration for Optimizing Sustained Memory Throughput in Heterogeneous HPC Devices

Syed Waqar Nabi and Wim Vanderbauwhede
School of Computing Science, University of Glasgow

Abstract. We are developing an optimizing compiler for scientific applications running on heterogeneous acceleration devices, which evaluates multiple design-variants in the search-space, based on estimates of parameters that affect performance. Sustained memory throughput is a key determinant. Here we present a synthetic micro-benchmark based on the stream benchmark, that enables design exploration of the non-trivial search-space of heterogeneous acceleration devices in general, and FPGAs in particular. We focus on its novelty of parametrizability features, and present results for four different devices to illustrate the utility of our approach in improving the memory performance for HPC.

Keywords: Optimization, Benchmark, Memory performance, Memory throughput, Bandwidth, FPGA, Heterogeneous, Stream, HPC, OpenCL

1 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. FPGAs are fundamentally different from instruction-processors as they can be configured repeatedly into customized circuits to suit the target application. While FPGAs have traditionally been used for prototyping or specific embedded applications, there is an on-going paradigm shift in the way FPGAs are programmed and used. Higher logic capacity and maturing High-level Synthesis (HLS) tools are pushing FPGAs into the mainstream of heterogeneous high-performance computing and big-data. With the advent of OpenCL-based HLS flows, as well as other software-friendly development approaches, FPGAs are increasingly amenable to programming by software designers.

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. The reason is that memory bandwidth is increasingly the performance bottleneck for many HPC applications. Out of thirteen dwarfs of HPC identified in [3], 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 [7]. Recently, its implementation for GPUs using CUDA and OpenCL was presented as GPU-STREAM [5]. FPGAs are still relatively new in the area of general-purpose and high-performance computing, and 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 gap.

In this paper, we present our extended version of the STREAM benchmark, called Multi-Platform-STREAM, or simply MP-STREAM. Developing an FPGA benchmark is a challenging task as FPGAs are a very different kind of target architecture when compared with CPUs and GPUs, and we wanted to make sure that we are exploring the entire design-space when we measure the memory-bandwidth. Our key contribution is in fact the introduction of a number of additional tuning parameters that have an impact on the sustained memory bandwidth. In this way, our benchmark can provide valuable insights to optimizing high-performance code for FPGAs.

The MP-STREAM benchmark is not limited to FPGAs, and is meant to be a generic heterogeneous benchmark. We consider such a benchmark is in-line with the increasingly heterogeneous nature of today’s HPC target machines. Hence we present results across four different architecture in this paper. FPGAs however remain our main focus.

We present our work by first referring to the original benchmark that consists of four kernels: copy, scale, add and triad. We briefly discuss GPU-STREAM, the adaptation of this benchmark for GPUs. We provide some information about the four heterogeneous targets that we use for the experiments presented here: one CPU target, one GPU target, and two FPGA targets. The execution flow of the benchmark is also presented. We then move on to what we consider is the main contribution of our work, which is the various generic as well as device-specific parameters that can be tuned to observe the behaviours of these heterogeneous devices under various conditions. This aspect of our work makes it distinct from previous works in this area, as our benchmark will now allow users to experiment with a larger design-space exposed by these tuning parameters.

Another important contribution of our work presented in this paper is the results from experiments on four heterogeneous targets, and some noticeable observations that arise out of it. We will discuss some 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.

2 Background and Requirement of a New Benchmark

STREAM [7] 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: \( a(i) = q \cdot b(i) \)
3. SUM: \( a(i) = b(i) + c(i) \)
4. TRIAD: \( a(i) = b(i) + q \cdot c(i) \)

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 [5] implements the four kernels listed above in OpenCL and CUDA frameworks for GPGPUs. The benchmark is open-source, and allows one to estimate the performance of a memory-bound kernel on GPUs. The sustained memory bandwidth of various GPU targets is reported against their peak performance. 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. The GPU-stream benchmark is in OpenCL but could not be effectively used for FPGAs either, 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. While the motivation for our benchmark was to explore the design-space for FPGA targets, the additional parameters we introduced provide a useful insight into memory performance on CPUs and GPUs as well, so our benchmark can be considered a multi-platform stream benchmark, or MP-STREAM.

3 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 [1], which is a heterogeneous parallel programming language and framework. We have tested the benchmark on four heterogeneous targets. The bandwidth is simply calculated as follows:

\[
\text{bandwidth} = \frac{N \cdot \text{words\_per\_array} \cdot \text{bytes\_per\_word}}{\text{time}_x}
\]

(1)

Where \( \text{time}_x \) is the minimum, maximum or average time for running the kernel over the entire array(s), giving us the maximum, minimum or average bandwidth. \( N \) is a constant scaling factor, that depends on how many words are accessed for each iteration of the kernel. The value of \( N \) for the copy, scale, add and triad kernels is 2, 2, 3 and 3 respectively.
3.1 Experimental Setup

Our experimental setup comprises two OpenCL-compatible FPGA targets, an Intel CPU and an Nvidia GPU (Table 1), demonstrating the portability of our benchmark across heterogeneous targets.

<table>
<thead>
<tr>
<th>Device</th>
<th>CPU</th>
<th>Intel Xeon CPU E5-2609 v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache size</td>
<td>10 MB</td>
<td></td>
</tr>
<tr>
<td>Peak Memory BW</td>
<td>34 GB/s</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>GPU</th>
<th>GeForce GTX Titan Black</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Memory BW</td>
<td>336 GB/s</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>FPGA-AOCL</th>
<th>Nallatech PCIe-385</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>Altera Stratix V GS D5</td>
<td></td>
</tr>
<tr>
<td>Peak Memory BW</td>
<td>25 GB/s</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>FPGA-SDACCEL</th>
<th>Alpha-Data ADM-PCIE-V7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>Xilinx Virtex 7 XC7</td>
<td></td>
</tr>
<tr>
<td>Peak Memory BW</td>
<td>10 GB/s</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Specifications of targets used for experiments.

3.2 Benchmark Parameters for Covering the Memory Design-Space

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. While these parameters provide an extensive cover of the memory-access architecture design-space, we are not yet claiming exhaustive coverage. We expect to keep updating and evolving this benchmark in response to feedback from the community, its execution on more targets, and the introduction of new compilers and optimizations for targets already evaluated.

Generic Design-Space Parameters

**Kernel** Our benchmark implements the same four kernels that were present in the stream benchmark: i.e., copy, scale, add and triad. While the stream benchmark would execute all four kernels in a single run, we have chosen to run only one kernel at a time, so that FPGA resources are exclusively used for that kernel.

**Source/destination of streams** The primary objective of this benchmark is to measure the sustained bandwidth to the device-DRAM (i.e. global/shared memory), as in the typical case this parameter determines the memory-roof. However, we have also included the ability to measure the sustained bandwidth of data streams directly between the host and the device (generally over PCIe).
Size of array The size of the array was the only controllable parameter in the original STREAM benchmark. Size has a significant impact on the bandwidth. Smaller array sizes have a proportionally larger memory latency impact, and to measure asymptotic performance we need to ensure the array size is large enough. Also, [7] notes that arrays have to be large enough to ensure that we are in fact measuring DRAM bandwidth in the cache-based targets.

Data type The benchmark currently supports integer and double types. In the case of copy kernel, this distinction is irrelevant from a computational perspective, but if we were to use double instead of integer type, this would translate into a memory-controller that coalesces 64-bits rather than 32-bits for read/write operations. For other kernels, there is obviously 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. Note that this does not necessarily imply SIMD processing of data inside the kernel.

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 [4, 6]. Effect of checking the effect of data access pattern was also suggested for the original STREAM benchmark, but never carried out as far as we know. In MP-STREAM, the arrays are organized as two-dimensional square matrices, and currently we test two patterns: contiguous data, and strided data, with fixed stride. For strided access, we do column-wise access of row-major data.

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 or threads, the total number of which is known as NDRange. Hence an NDRange kernel looks something like this:

```c
//** HOST **/
const size_t globalSize [] = {ARRAY_SIZE, 0, 0 };  
clEnqueueNDRangeKernel ( kernel , globalSize , ... );

//** DEVICE **/
__kernel void streamKernel (...){
```
Single work-item, flat-looping kernel At the other extreme, we can launch just one work-item, and have a conventional for loop in the kernel performing for covering the iteration-space, e.g.:

```c
index = get_global_id(0);
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:

```c
const size_t globalSize[] = {1, 0, 0};
cEnqueueNDRangeKernel (kernel, globalSize, ...);

__kernel void streamKernel(...) {
    for (index=0; index < ARRAY_SIZE; index++)
        c[index] = a[index];
}
```

Loop unroll factor This optimization reduces the branch penalties and control overhead, and can be controlled in OpenCL through the __attribute__((opencl_unroll_hint(n))) command which specifies unrolling by a factor of n. Our benchmark’s build scripts generate custom kernel code with these optimizations as specified.

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 Specific Design-Space Parameters FPGA device vendors often have optimization techniques that are not necessarily part of the OpenCL standard. For example, the AOCL compiler has a number of optimization parameters discussed in [2]. The following optimization parameters were relevant to this benchmark:

- Number of SIMD work-items
- Number of compute-units
Similarly, 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:

- Pipeline loop
- Pipeline work-items
- Maximum memory ports
- Memory port data width

## 4 Experiments in Design-Space Exploration using MP-STREAM

In subsection 3.1, we presented the four targets that we have used for running experiments based on the MP-STREAM benchmark. The main 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. While we gained some interesting insights into various target devices in our experiments, exhaustive testing or aggressive optimization across all targets is not the intent of this work. We would thus like to emphasize that there is no guarantee we have achieved the best possible results on the tested targets. We hope the availability of our benchmark will allow users to test different parameter combinations based on their own domain knowledge of various architectures, and share their recommendations with the larger community.

### 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. Another observation is that the GPU not only has a much higher peak bandwidth than the FPGAs (the horizontal dotted lines), but that we get sustained bandwidth very close to this peak at large array sizes. The FPGA targets are relatively further away from their peaks, but if we look at Figure 1(b) where we use vector data types in the kernel signatures – effectively coalescing memory accesses – then we see the FPGA targets approaching their peak bandwidths. It is interesting to note that vectorization shows little improvement and even has a detrimental effect on the memory bandwidth for CPU as well as the GPU. Also, note that for CPU at small array sizes, we may be seeing the effect of caching, since we run the same kernel multiple times on the same data.
Effect of Data Contiguity

The effect of data contiguity on sustained memory bandwidth to a DRAM is well known. We ran the MP-STREAM benchmark on all four targets, varying the array size as well as testing two patterns of access: contiguous, and strided with a fixed stride. For strided access, we accessed the row-major D array in a column-major fashion, which means the size of stride is equal to the square-root of the array-size in all cases. The results are shown in Figure 2(a). 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 once at the beginning to convert subsequent strided accesses to contiguous accesses.

Another interesting observation is that, if we focus only on strided accesses, then after an initial improvement in performance with increasing array (and stride) size, the CPU and GPU start seeing a decline in performance as we increase the array (and stride) size. Keeping in mind that the axis are log-scale, it can be noted that the decline is significant. FPGAs have no such trend. For the CPU, we see a sharp decline at an array size of around 10MB, which is the size of the cache in this CPU.
As discussed earlier, we tested three different ways of expressing the main array access loop. The result of testing all these methods on the four targets is presented in Figure 2(b).

As could be expected, the GPU and CPU show the best performance for the NDRange Kernel, which means that 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 1-work-item kernel, and the loop managed explicitly at the kernel. While the AOCL target seems to be relatively unaffected, 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 anticipate that experimenting with more devices will show similar target-specific characteristics, which supports our argument that a route to exposing and then 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 3(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 to the native OpenCL feature of using vectorized data types as the kernel arguments. As we can see from the results in Figure 3(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, have less consistent results, eventually giving poorer performance as we increase their scale. While the results are not shown here, we also found that the AOCL optimizations take up more FPGA resources when compared with equivalent native OpenCL optimizations. We feel this 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**

We have made some observations earlier specific to each experiment, but we would also like to highlight some generic observations we made in the course of writing and experimenting with this synthetic benchmark.

An obvious 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, 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 these heterogeneous targets more accessible, which is what our goal is with the TyTra framework. Both the manual and automated 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 and choices.

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-formatting 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.

### 5 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 even the recently proposed GPU-STREAM benchmark as there are a number of 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 presented too. 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 will make this benchmark publicly available, and plan to have a website for users to contribute and share their results with the wider community.

Acknowledgement The authors acknowledge the support of the EPSRC for the TyTra project (EP/L00058X/1).

References