GPU-STREAM: Benchmarking the achievable memory bandwidth of Graphics Processing Units

Tom Deakin* and Simon McIntosh-Smith†
Department of Computer Science
University of Bristol
Bristol, UK
Email: *tom.deakin@bristol.ac.uk, †simonm@cs.bris.ac.uk

***Abstract*—Many scientific codes consist of memory bandwidth bound kernels — the dominating factor of the runtime is the speed at which data can be loaded from memory into the Arithmetic Logic Units. Generally Programmable Graphics Processing Units (GPGPUs) and other accelerator devices such as the Intel Xeon Phi offer an increased memory bandwidth over CPU architectures. However, as with CPUs, the peak memory bandwidth is often unachievable in practice and so benchmarks are required to measure a practical upper bound on expected performance.**

We present GPU-STREAM as an auxiliary tool to the standard STREAM benchmark to provide cross-platform comparable results of achievable memory bandwidth between multi- and many-core devices.

I. Measuring Memory Bandwidth

The STREAM Benchmark [1] measures the time taken for each of four simple kernels to be run (α is a scalar constant):

1) Copy: $c[i] = a[i]$
2) Multiply: $b[i] = \alpha c[i]$
3) Add: $c[i] = a[i] + b[i]$
4) Triad: $a[i] = b[i] + \alpha c[i]$

It is simple to calculate how many bytes each of these kernels requires to be read from and written to memory, assuming perfect caching. Let β be the number of bytes to represent one element — for double precision $\beta = 8$. For an array of length N, copy and multiply move $2N\beta$ bytes, and add and triad move $3N\beta$ bytes. The ordering of the kernels ensures caches are invalidated, and as long as the arrays are large enough the data must be reloaded from main memory. The achieved sustained memory bandwidth can be found by dividing these numbers by the time to execute the corresponding kernel.

GPU-STREAM implements these kernels in both the OpenCL and CUDA programming frameworks. This allows the benchmark to be used across a wide range of hardware from a wide range of vendors.

II. Related work

The data arrays remain resident on the device for the lifetime of the timing measurements. This is in contrast to memory bandwidth tests provided in the GPU vendors' compute SDKs and SHOC [2] which include transfer time over the PCIe bus — something which in scientific codes is typically avoided for the compute kernels themselves. These examples do not allow the programmer to draw accurate conclusions as to how well their memory bandwidth bound compute kernel is performing for the given hardware. GPU-STREAM addresses this issue.

Likewise, the benchmark clpeak [3] complicates the bandwidth measurement by considering performing a reduction of a global buffer using various OpenCL vector types — this is not at all a comparable metric to STREAM. Additionally, this approach does not align with recommended reduction techniques on GPUs [4].

III. Portable results

We will demonstrate that both STREAM and GPU-STREAM obtain around 80% of theoretical peak of the appropriate hardware, and so together allow a simple, fair and valid comparison of memory bandwidth on a graph such as that in Figure 1.

IV. Error Correcting Code Memory

Error Correcting Code (ECC) Memory is used to account for single and double bit errors in the memory system, and works by adding a parity byte for every eight data bytes. On GPUs this is all handled directly in the hardware, using the same number of memory channels. Firstly, this reduces the total available memory size by 12.5% as every byte requires one extra bit. Secondly, the parity byte must be checked and the memory corrected. This means we can also expect to lose at least 12.5% of the peak memory bandwidth as we must read an extra byte for every eight byte read request.

Typically CPUs have no control about using ECC memory as this is handled by the DRAM memory controller. GPUs can toggle this setting in the driver.

GPU-STREAM achieves a similar percentage of peak memory bandwidth whether ECC is turned on or off. Similarly, the difference between achieved memory bandwidth is approximately 12.5%, indicating that the hardware checking of the memory does not significantly affect the bandwidth compared to the extra data movement required as a result of ECC — moving more memory is more harmful than the parity check logic as valuable memory bandwidth is being used.

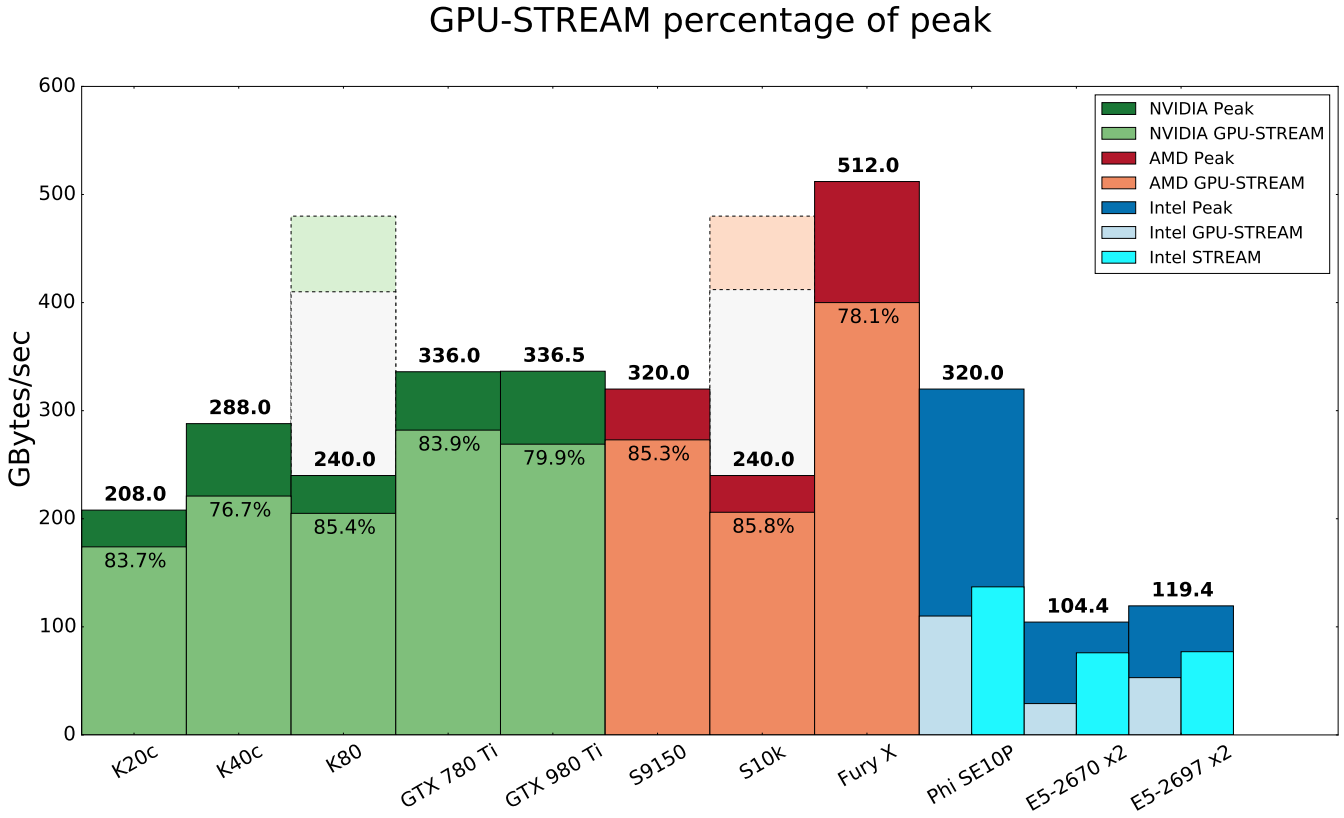

Fig. 1. GPU-STREAM results

V. Conclusions

"How well is my memory bandwidth bound kernel performing?"

For multi-core CPUs this question is answered easily by comparing your achieved bandwidth to that reported by STREAM. GPU-STREAM allows the same comparison to be drawn for GPUs.

Note that GPU-STREAM does not implement any tuned versions of the kernels, which is allowed under the STREAM rules. A standard technique of allocating multiple array elements to single processing elements, as recommended by vendor best practices guides, reduced the performance of the benchmark. This demonstrates that the runtimes have matured and are scheduling the work on the device's compute units in a performant way, without programmer intervention. This highlights the case that OpenCL can provide performance portable code.

GPU-STREAM is Open Source and available on GitHub at `github.com/UoB-HPC/GPU-STREAM`. The webpage maintains a repository of all our results and we encourage submission of results.

References

[1] 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.

[2] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable Heterogeneous Computing (SHOC) Benchmark Suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU-3. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1735688.1735702

[3] K. Bhat, "clpeak," https://github.com/krrishnarraj/clpeak, 2015. [Online]. Available: https://github.com/krrishnarraj/clpeak

[4] AMD, "OpenCL Optimization Case Study - Simple Reductions," http://developer.amd.com/resources/documentation-articles/articles-whitepapers/opencl-optimization-case-study-simple-reductions/.