Matrix-Matrix Multiplication for BOINC with SciGPU-GEMM

R. G. Edgar,[1,2] R. Olivares-Amaya,[3] L. Vogt,[3] M. A. Watson[1,3] & A. Aspuru-Guzik[3]

[1]Initiative in Innovative Computing, Harvard University,
[2]School of Engineering and Applied Sciences, Harvard University,
[3]Department of Chemistry and Chemical Biology, Harvard University

rge21@seas.harvard.edu

http://scigpu.org/

The Need for SciGPU-GEMM

Increasing the efficiency and reducing the cost of solar cells are keys to the increased adoption of solar power. There is a bewildering variety of candidate materials for solar cell construction, and we are using the distributed computing power of the BOINC project to guide material selection via quantum chemistry calculations. Many of these calculations spend a considerable amount of time performing matrix-matrix multiplications, which are easily accelerated with GPUs with CUBLAS. The presence of CUDA-capable GPUs on many BOINC client machines opens up the tantalising prospect of using CUDA acceleration, but there are two key obstacles:

1. Small device memory on consumer-level GPUs
2. Lack of double precision hardware on the same GPUs

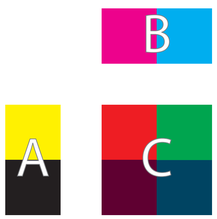
We wrote the `SciGPU-GEMM` library to bypass these difficulties.

Cleaving GEMMs

Consider the matrix multiplication

$$C = A \cdot B$$

We can divide A into a column vector of matrices and B into a row vector of matrices - we refer to this process as *cleaving* the matrices. The matrix C can then be assembled from the multiplication of each pair of sub-matrices from A and B.



In `SciGPU-GEMM`, the matrices are automatically cleft according to the amount of memory available on the current GPU. Each pair of sub-matrices is staged through the GPU and multiplied using the CUBLAS library. The arguments for a call to the matrix cleaver are identical to those for a standard call to `SGEMM` or `DGEMM`. Figure 1 shows how the cleaver performs on our cluster. When the GPU is limited to using only 16 MiB of RAM, clear 'steps' appear in the time taken, corresponding to points where extra cleaving became necessary.

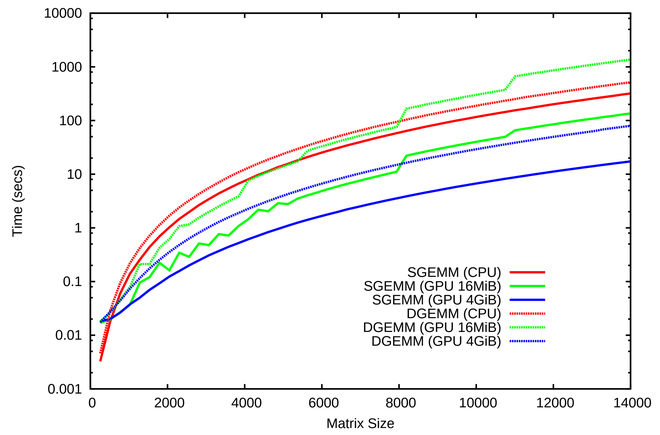


Figure 1: *Performance of the matrix cleaver, as a function of (square) matrix size. Timings for both single and double precision are shown, for an Intel 3 GHz 'Harpertown' Xeon and an NVIDIA Tesla C1060. For the Tesla, we show timings where the card was limited to using 16 MiB of RAM, in addition to using all of its available RAM. For both the single and double precision cases, this caused cleaving to begin at around a matrix size of 1024. All timings are averaged over 20 calls.*

Heterogeneous Computing with MGEMM

Many of our calculations require intermediate numerical precision: single precision is generally sufficient, but double precision is sometimes required. For such calculations, we have written a mixed-precision matrix multiply routine for `SciGPU-GEMM`. This splits the matrices into 'large' and 'small' components

$$C = \left(A^{\text{large}} + A^{\text{small}}\right) \cdot \left(B^{\text{large}} + B^{\text{small}}\right) = \underbrace{AB^{\text{large}} + A^{\text{large}}B^{\text{small}}}_{\text{CPU}} + \underbrace{A^{\text{small}}B^{\text{small}}}_{\text{GPU}}$$

The $A^{\text{small}}B^{\text{small}}$ term is run in single precision on the GPU (cleaving if needed). The other two terms are run in double precision on the CPU. However, each of the 'large' matrices should be sparse, cutting the computational complexity significantly. To split the matrices, we define a cut-off value, δ. If $|A_{ij}| > \delta$, that element is considered 'large.' We call our routine `MGEMM`, for 'multi-precision general matrix multiply.' It operates similarly to the other `*GEMM` routines, but takes δ as an extra argument.

Figure 2 shows `MGEMM` performance on a real quantum chemistry calculation. There is a clear range of δ values where `MGEMM` is more accurate than `SGEMM` while also outperforming `DGEMM` on the CPU.

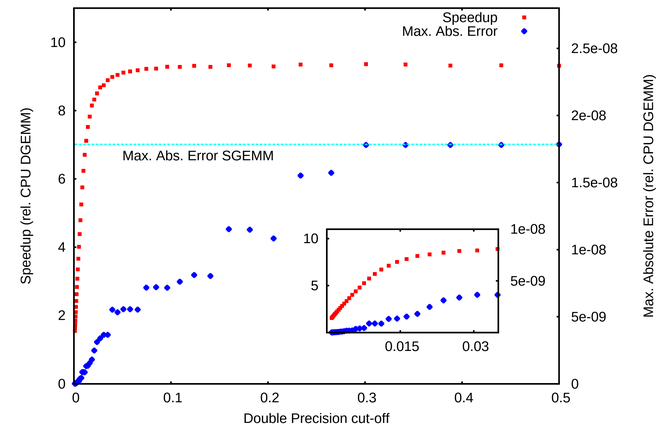


Figure 2: *Performance of `MGEMM`, as a function of the cut-off value, δ, for a pair of matrices (sizes 4485×4186 and 4186×4485) from a quantum chemistry calculation of the taxol molecule. Left ordinate: Time taken for various `GEMM` calls (Intel 3 GHz 'Harpertown' against NVIDIA Tesla C1060). Right ordinate: Maximum absolute error in a single element. The subplot magnifies small δ values, showing where `MGEMM` offers improved performance.*

Conclusion

The `SciGPU GEMM` library solves two problems associated with using GPUs for matrix-matrix multiplication in a distributed environment such as BOINC: a lack of available memory, and the absence of double precision hardware. The library (with full Doxygen documentation) is available for download from `http://scigpu.org/`.

Acknowledgements

The authors acknowledge financial support provided through NSF Award PHY-0835713. Harvard University has been named a CUDA Center of Excellence by NVIDIA, and has received several hardware donations which were used in this work.