GPU Programming 2017/18

Projects

Matrix transpose

- Develop an efficient CUDA program that computes the transpose of a matrix for arbitrary data types (with possibly different size).
- Your program should be efficient irrespective of the matrix size and the data type used.
- Analyse if your implementation achieves the performance that is predicted by the theoretical tools discussed in the lecture.
- Compare your performance to existing library implementations.

Matrix-matrix multiplication

- Develop an efficient CUDA program that computes the product of two, not necessarily quadratic matrices.
- Your program should be efficient irrespective of the matrix size and the data type used.
- Analyse if your implementation achieves the performance that is predicted by the theoretical tools discussed in the lecture.
- Compare your performance to existing library implementations.

Merge sort

- Develop an efficient merge sort in Cuda.
- Explore if grid synchronization can be used to improve the performance.
- Analyse if your implementation achieves the performance that is predicted by the theoretical tools discussed in the lecture.
- Compare your performance to existing library implementations.

Quick sort

- Develop an efficient merge sort in Cuda.
- Explore if dynamic parallelism can be used to improve performance.
- Analyse if your implementation achieves the performance that is predicted by the theoretical tools discussed in the lecture.
- Compare your performance to existing library implementations.

Placement-`new` for CUDA memory management

- Implement a memory arena and placement-`new` that enables to easily transfer a set of instance from the host to the device (e.g. for a raycaster where one has various geometric primitives compromising the scene). The data transfer to the device and the data access there should be as simple and transparent as possible.
- Compare the performance of your approach to Cuda's unified memory model.
- The classic book by Alexandrescu [Ale01, Ch. 4] is a good reference on placement new with pointers to other literature.

Octree construction

- Implement an octree that provides a hierarchical data structure for a geometric data set (in the simplest case a set of randomly generated points with closest point queries as objective).
- Explore the use of dynamic memory allocation on the device during construction (and potentially dynamic parallelism).
- A good reference on acceleration structures is the book by Pharr and Humphreys [PH10].

Fast Fourier Transform

- Implement an efficient Fast Fourier Transform in CUDA.
- Employ your Fast Fourier Transform to implement audio or image processing, e.g. denoising.
- Explore the use of lower precision data types (e.g. fp16, int8) to speed up computations.
- Compare the performance of your implementation to those of the cuFFT library.

Tone Mapping using histograms

- Implement an efficient histogram computation in CUDA.
- Use the histogram to implement tone mapping.[1]
- Explore the use of lower precision data types (e.g. fp16, int8) to speed up computations and the direct visualization of the result using the CUDA-OpenGL interoperability.

N-Body simulation

- Implement an efficient N-Body simulation in CUDA.
- Explore the use of different potentials to describe the interaction between particles / bodies, e.g. a graviational Newton potential or van der Waals forces.
- Visualize the result using the CUDA-OpenGL interoperability.

Image segmentation using clustering

- Implement an efficient k-nearest neighbors algorithm in CUDA.
- Use your k-nearest neighbors implementation to determine image segmentations.[2]

Convolution

- Implement an efficient 2D convolution in CUDA.
- Test your convolution for image denoising; in particular, explore alternatives for the filter.
- Explore if the computations can be speeded up by using lower precision data types (e.g. fp16, int8) in CUDA.

Wavelet-based image compression

- Implement a fast wavelet transform for images for non-standard separable wavelet bases. A popular choice for the filter coefficients is the family of Daubechies wavelets. Begin by implementing a 1D wavelet transform.
- Use your implementation of the fast wavelet transform for image compression.

---

[1] See for example `http://resources.mpi-inf.mpg.de/departments/d4/teaching/ws200708/cg/slides/CG13-ToneMapping.pdf`.

[2] See e.g. `https://de.mathworks.com/help/images/examples/color-based-segmentation-using-k-means-clustering.html?requestedDomain=www.mathworks.com`.

- Explore the use of dynamic parallelism to speed up computations.
- An accessible book on wavelets is [SDS96] as well as the related course notes [SDS95b]; [SDS95a]. More details can, for example be found in Mallat's book [Mal09] and Daubechies' classic text [Dau92].

References

[Ale01] A. Alexandrescu. *Modern C++ Design*. Addison Wesley, 2001.

[Dau92] I. Daubechies. *Ten Lectures on Wavelets*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.

[Mal09] S. G. Mallat. *A Wavelet Tour of Signal Processing: The Sparse Way*. third ed. Academic Press, 2009.

[PH10] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. second. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[SDS95a] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. "Wavelets for Computer Graphics: A Primer, Part 2". In: *IEEE Computer Graphics and Applications* 15.4 (1995), pp. 75–85.

[SDS95b] E. J. Stollnitz, T. DeRose, and D. H. Salesin. "Wavelets for Computer Graphics: A Primer, Part 1". In: *IEEE Computer Graphics and Applications* 15.3 (1995), pp. 76–84.

[SDS96] E. J. Stollnitz, T. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.