Particle-Mesh Methods on Graphics Processing Units: The Application of Emulated Double precision to Improve Speed and Precision

Otonyo Mangete

Department of Physics, Drexel University

Introduction

The Particle-Mesh (PM) method is used regularly to simulate gravitational systems such as stars, galaxies and dark matter distributions in cosmological simulations.

The Fast Fourier Transform (FFT) is integral to the PM method and its accuracy and speed are vital in order to use it successfully.

The Nvidia GPU is emerging as an low-cost, high-speed computational platform and there is some interest in the scientific community in applying GPU systems to the PM system

Aim

We will tackle the critical part of the PM method, the FFT, first. The Nvidia GPU system provides a FFT library called CUFFT which operates well only in single precision and is suboptimal with respect to the GPU's resources in both double and single precision

The lack of double precision performance is due to the design of the GPU itself. The GPU is a single-precision device and computes with only a fraction of its resources in double precision.

We address this problem by computing in Emulated Double Precision (EDP). This method computes an“almost” double-precision floating point number using two single-precision floating point numbers . This implies that we use four single-precision floating point numbers to represent one complex number.

Method

For our preliminary work on the EDP-FFT, an optimized one-dimensional FFT is tested. The optimized FFT is from work by Volkov et al (1) who have improved upon the performance of the Nvidia FFT by a factor of 3 at least.

This optimized FFT is of the Cooley-Tukey type and has radix sizes 2, 4, 8 and 16. We compute (following Volkov et al) transform sizes 8, 16, 64, 256, 512 and 1024.

A small library of EDP functions by Bailey (2) is ported to the Nvidia CUDA system and is used to compute the EDP FFT for the given sizes. The FFT's are executed in batches for timing purposes and each transform size has a total of more that 16 million elements for each transform.

We also compute single-precision FFT's with Nvidia's CUFFT and Volkov's optimized FFT for comparison. We compare speed-up and memory usage of the GPU as well. This is in keeping with tests carried out by Volkov et al (1) on their FFT.

We were unable to get a double-precision GPU for testing purposes so we used the best-case scenario for the double precision CUFFT code. This implies that DP will be slower by a factor of 8 at best since this is the ratio of DP to SP processing elements.

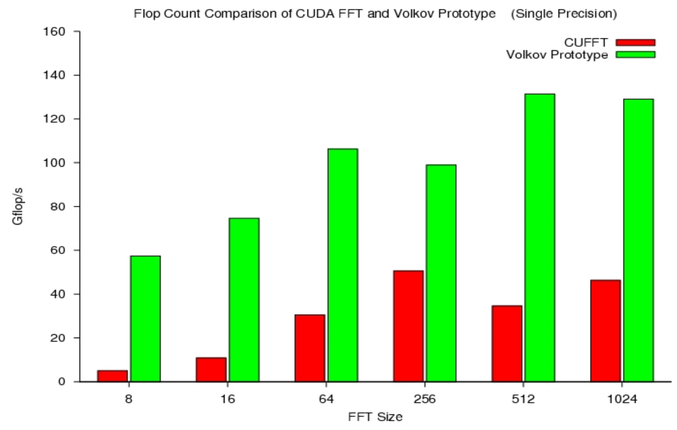

Figure 1: Flop Count of FFT Computation for various sizes using single precision FFT code from Nvidia CUFFT and Volkov's optimized 1-D FFT

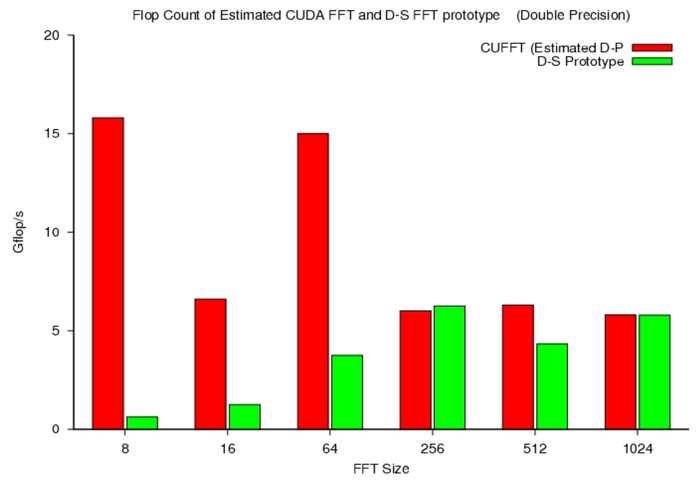

Figure 2: Flop Count of FFT computation for various FFT sizes using the EDP-FFT Prototype code and an estimated DP CUFFT .

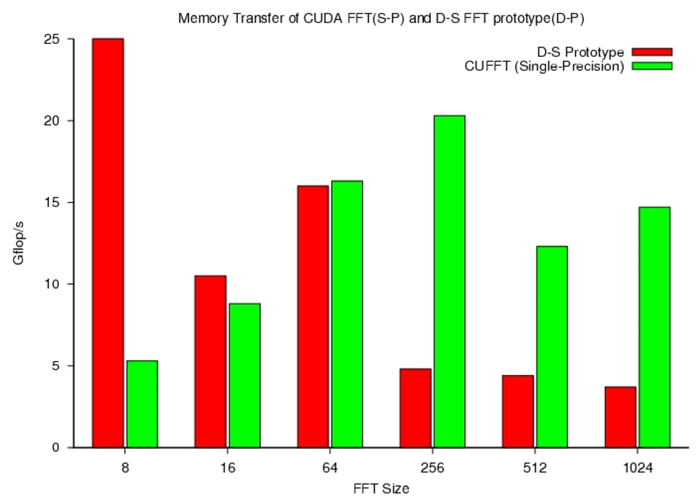

Figure 3: Memory Transfer Comparison between CUFFT and D-S prototype for various FFT lengths..

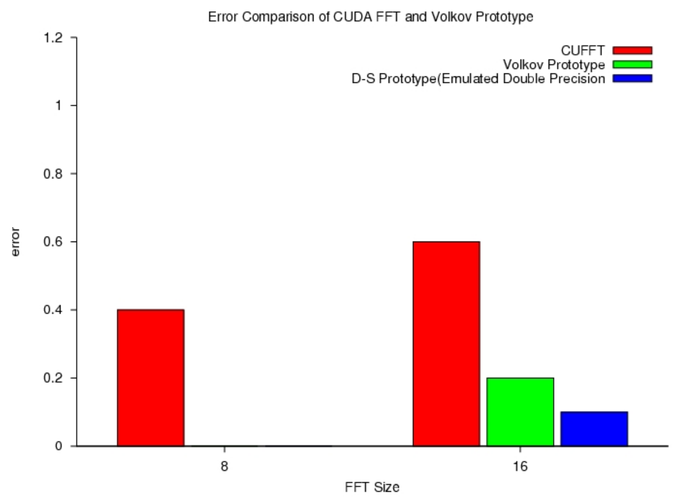

Figure 4: Error comparison between CUFFT, Volkov prototype (both in single precision) and D-S prototype.

Results

Figures 1-4 show preliminary results from the application of Double-Single precision to the FFT algorithm. From Figure 1, it is clear that the Volkov FFT routine is superior to the Nvidia FFT routine.

We also see from figure 2 that we good performance with batched 1-D FFT's for sizes 8 and 64. The performance of sizes 16, 256, 512 and 1024 are much worse and a cause for concern. From Figure 3, we see that memory transfer is also bad, especially for sizes 256, 512 and 1024.

The reason for the sudden drop in flop count for the D-S prototype is simple: The optimized FFT in emulated DP requires many more function calls and these will have to be further optimized. This is done in part for the single-precision length-64 FFT and explains the 'jump' in performance for that particular FFT. The length-8 FFT fits into the memory bank of the FFT and can be computed fast despite memory transfer problems.

From Figure 4, we can see that the errors accumulated for the FFT's (for the part-optimized length-16 and also the length-8 FFT) are better in the D-S formulation than in CUFFT.

We also note that the Cooley-Tukey algorithm requires computationally expensive bit-reversal routines and this will take a toll on speedup for larger transform sizes.

Conclusion

From the results above, it appears that the emulated double precision system provides an viable alternative while using GPU's for double-precision dependent calculations. It is clear that even more care is required to design optimal FFT libraries for emulated double-precision than for just single precision-based libraries.

Optimal FFT's in single precision have been computed for one and two dimensions by Govindaraju et al(3) and even for three dimensions by Nukada et al(4) . The 3-D FFT of Nukada shows drastic degradation in performance when double precision is used on the Nvidia GPU (Nukada – private communication).

Porting optimal FFT algorithms for the GPU such as these to Emulated Double Precision for use in a Particle-Mesh code is one of the goals of our project.

REFERENCES

1. Volkov et al 2008

2. Bailey, D.H – DSFUN library for Double-Single precision - http://crd.lbl.gov/~dhbailey/mpdist/

3. Govindaraju et al – High-Performance Discrete Fourier Transforms on GPUs – (at download/microsoft.com

4. Nukada et al – Bandwith-Intensive 3-D FFT kernel for GPUs using CUDA – SC2008 paper

Acknowledgements

I would like to thank Steve McMillan, Evghenii Gaburov and Ernie Mamikonyan for their assistance with this project.