CUDA-Based GPU Computing Framework for GNU Octave

Inspired by Jacket from AccelerEyes - GPU Engine for Matlab®*

Jaideep Singh, Indian Institute of Technology Roorkeee, INDIA

Introduction

This poster presents the design of a CUDA-based parallel processing framework for GNU Octave[1]. GNU Octave is a high-level interpreted language, primarily intended for numerical computations. GNU Octave being an open source alternative to Matlab®, is widely used in academic and research institutes. This work is inspired by the design and functionalities of Jacket[2], a GPU Engine for Matlab®. Introduction of new GPU types helps to avoid any data transfers over PCIe in moving from one GPU routine to another. To my knowledge, this is the first attempt to build a GPU framework for Octave, contrary to previous attempts to provide GPU variants for a set of Octave functions.

History of Parallel Computing in Octave

• The development of a complete interface to the main parallel programming libraries for Octave had been never accomplished [3] before MPI Toolbox for Octave (MPITB). MPITB[4] allows Octave users to build their own LAM/MPI based parallel applications.

• After the introduction of CUDA technology in 2006, there have been some attempts to enable GPU computing in Octave using CUDA, which can be found in Octave mailing list [5].

• This has been limited to providing a plug-in with a set of GPU accelerated routines for commonly used Octave functions

• This approach suffers a serious drawback as it incurs a data transfer over PCIe (limited to 8 GB/s in PCI x16 generation) between CPU and GPU memory in every GPU function call which severely limits the performance of this design.

Advantages & Capabilities of the GPU Framework

• The framework allows Octave users to accelerate their software written in Octave high-level **'M'** language on GPUs with ***minimal code modifications***. After casting data into ***gpuTypes***, the same code gets accelerated many times on the GPUs.

• The framework allows users to build wrappers around CUDA-based high performance GPU libraries like nVidia's **cuBlas**, **cuFFT** and other 3rd party libraries for **accelerating BLAS and linear algebra routines on GPUs**.

• Data that resides in GPU memory can be visualized directly using CUDA-OpenGL interoperability, avoiding any data movements and can be used to run **visual simulations at accelerated speeds**.

• The framework scales computations to multiple GPUs in the system. It provides the user with the option of selecting a particular device for execution and synchronization functions.

Design Approach

• GNU Octave is written in C++ and supports extensions on itself, by the use of dynamically loaded modules, and shared libraries.

• Object hierarchy is supported in GNU Octave with the help of a type system. We can inherit a type from **octave_value**, the canonical holder, and implement its virtual functions so that we have a new type to work with.

• When the custom class (which inherits from **octave_value** class) describing the new data type is compiled into a shared object (**.oct**), the symbols are exported into the library without linking to the octave library.

• Octave searches and loads the *DEFUN_DLD* functions defined in custom class from the **.oct** file and invokes the same with the arguments.

• In our case, we inherit from **octave_value** and introduce new data types, termed as **gpuTypes** in this poster, which hold data in GPU device memory and can be passed to GPU functions for GPU-based processing by launching **kernels** or calling GPU libraries.

• Arithmetic and logical operators are overloaded, which perform intuitive functions on the object from the interpreter itself.

• Octave v3.2.3 came with OOP support. Octave users can now create custom classes and overload functions which are given precedence over the generic versions by the Octave runtime.

• Octave runtime searches for the functions definitions based on the parameter list and thus this method can be used to overload Octave built-in routines.

Implementation of CUDA-GPU Framework

• A new data type, **gFloat** is introduced into the Octave runtime by building a custom C++ class that extends from **octave_value**.

• Since the Octave interpreter recognizes the **gFloat** data type, we can define member functions for the **gFloat** class which can launch **CUDA kernels** to perform computation on the **gFloat** class objects on the GPU. The various logical and arithmetic operators are overloaded for the **gFloat** class.

• All the routines that operate on **gFloat** class are prefixed with **'gpu'** as the Octave interpreter calls the generic implementations of built-in routines rather than class member routines.

• This ***limitation*** is removed by implementing a wrapper class over **gFloat**, viz. **gpuFloat** using Octave OOP features, which allows us to overload Octave built-in routines like the mathematical functions; e.g. exp, log and many others as shown below.

```
% gpuFloat class constructor
function out = gpuFloat( in )
    % out.data holds gFloat object and can be used in
    % member functions to perform arithmetic
    if ( nargin == 1 )
        if( strcmp( class(in),'gFloat'))
            out.data = in;
            out = class (out, "gpuFloat");
            return;
        endif
        % Make copy of CPU vector/matrix in GPU memory
        if ( isreal (in))
            out.data = gsingle( in );
            out = class (out, "gpuFloat");
        else
            out.data = [0];
            out = class (out, "gpuFloat");
        endif
    endif,
end
```

Profiling GPU Framework For Performance

For profiling the GPU framework developed for Octave, two samples hosted on the AccelerEyes site, viz., ***Monte-Carlo Simulation of Pi [6]*** and ***Black-Scholes Financial Computation*** were used. The code is written in Octave 'M' language. After casting the variables to **gpuTypes**, the same code gets acceleration on the GPU with no extra programming effort.

```
% Number of trials : CPU Version
for i=1:5
NSET = number_of_trials(i);

X = single( rand( 1, NSET ) );
Y = single( rand( 1, NSET ) );


tic;
distance_from_zero = sqrt( X.*X + Y.*Y );
inside_circle = (distance_from_zero <= 1);
pi_result = 4 * sum(inside_circle) / NSET;
t_result = toc;
end
```

```
% Number of trials : GPU Version
for i=1:5
NSET = number_of_trials(i);

X = gpuFloat( rand( 1, NSET ) );
Y = gpuFloat( rand( 1, NSET ) );
gsync();        % Synchronize with CPU

tic;
distance_from_zero = sqrt( X.*X + Y.*Y );
inside_circle = (distance_from_zero <= 1);
pi_result = 4 * sum(inside_circle) / NSET;
t_result = toc;
end
gsync();
```

Monte-Carlo Estimation of PI Benckmark*

Number of Trials	CPU_time(sec)	GPU_time(sec)	Speedup
131072	0.002070	0.002507	**0.825773**
524288	0.020664	0.007324	**2.820955**
1048576	0.056381	0.017415	**3.237888**
8388608	0.164864	0.037373	**4.050907**
16777216	0.326206	0.073764	**4.422301**

Black-Scholes Financial Computation Benchmark*

Input Data Size	CPU_time(sec)	GPU_time(sec)	Speedup
24000x1	0.178541	0.059402	**3.00564**
64000x1	0.472425	0.063777	**7.40745**
104000x1	0.776785	0.072706	**10.6839**
164000x1	1.235121	0.125247	**9.86148**

***System Specifications :**
CPU: **Intel® Core™ i7-2630QM CPU**, 6M Cache, 2.00 GHz with 6 GB RAM
GPU : **nVidia GeForce GT 540M**, 96 CUDA cores @ 1.344 GHz, nVidia driver *v270.41*, *Octave v3.2.3*, *CUDA 3.2*

Conclusion And Future Work

The GPU framework presented in this poster allows Octave users to leverage massively parallel CUDA cores to accelerate Octave processing. The benchmarks show that this framework is capable of accelerating applications written in Octave M-language with no code modifications. The GPU Octave framework is completely transparent to the user and can be extended easily to become more useful to the Octave user commuting. **The framework is designed such that it can be used by Octave users to transfer computations onto the GPUs with no prior experience in CUDA or GPGPU in general**.

Future course involves building more GPU classes and routines to make the framework more generic. Graphics support using OpenGL to visualize GPU data on the fly while performing simulations can be easily added to this framework. The emergence of GPU math-libraries like **libJacket** from AccelerEyes, which provides a huge set of mathematical routines, can be easily integrated into this framework.

References :

[1] Octave, http://www.gnu.org/software/octave/.

[2] AccelerEyes,GPU Computing with Matlab®, Python, C, C++, Fortran, www.accelereyes.com .

[3] J.W. Eaton, J.B. Rawlings,"Ten years of Octave – Recent developments and plans for the future", in DSC 2003 Proceedings of the 3rd Int.Wshp. on Dstr.Stat.C, March 2003, Vienna Austria.

[4] MPITB for Octave: Octave Parallel Computing with LAM/ MPI and Open-MPI http://atc.ugr.es/~javier/mpitb_old.html

[5] P. Kienzle, et al. "Octave-Forge repository": http://octave.sourceforge.net/, May, 2011, announcement seen in http://www.octave.org/octave-lists/archive/octave-sources.2001/msg000 10.html , Oct, 2001.

[6] Jacket Application Examples- http://www.accelereyes.com/support/application_examples