GPU accelerated curve fitting with IDL

Michael D. Gallo, mgalloy@txcorp.com
Tech-X Corporation

Work supported by NASA SBIR Grant #NNX12CA79C IN33A-1529

Introduction

Abstract

Curve fitting is a common mathematical calculation done in all scientific areas. The Interactive Data Language (IDL) is also widely used in this community for data analysis and visualization. We are creating a general-purpose, GPU accelerated curve fitting library for use from within IDL.

We have developed GPULib, a library of routines in IDL for accelerating common scientific operations including arithmetic, FFTs, interpolation, and others. These routines are accelerated using modern GPUs using NVIDIA's CUDA architecture. We will add curve fitting routines to the GPULib library suite, making curve fitting much faster.

In addition, library routines required for efficient curve fitting will also be generally useful to other users of GPULib. In particular, a GPU accelerated LAPACK implementation such as MAGMA is required for the Levenberg-Marquardt curve fitting and is commonly used in many other scientific computations. Furthermore, the ability to evaluate custom expressions at runtime necessary for specifying a function model will be useful for users in all areas.

Current capabilities

The features of GPULib include:

- Accelerated basic vector, as well as matrix, arithmetic for float, double, complex, and double complex arrays
- Accelerated interpolation
- Accelerated FFT (1-, 2-, and 3-dimensional, as well as batches)
- Accelerated special functions like LGAMMA
- Accelerated common IDL routines like HISTOGRAM and WHERE
- Accelerated special purpose image processing operations like the Radon transform
- Array indexing and efficient subarray operations
- Use of streams to hide memory transfer times
- Memory transfer and allocation
- Execute using pure IDL when CUDA-enabled hardware is not present

GPULib includes demos in the areas of:

- Deconvolution in the image processing
- Spectral angle mapping
- Finite-different time-domain (FDTD) 3-dimensional reconstruction

As well as demos of advanced GPULib features:

- New operator overloading API
- Using page locked memory to improve memory transfer
- Using streams to improve performance by doing asynchronous calculations/memory transfers

Future work

OpenCL

Use of the open standard OpenCL implemented by several hardware vendors from within GPULib is an exciting prospect with several advantages:

- OpenCL is an open standard and runs on multiple vendor hardware implementations including both CPU and GPU hardware.
- OpenCL uses runtime compilation of kernels allowing easy construction of custom kernels at runtime (useful for creating user-defined fitting functions)

The disadvantages of using OpenCL are also significant, but are being improved.

- Current library support for useful GPULib functionality such as FFTs, BLAS, and LAPACK is less mature in OpenCL than the CUDA version. There has been a clMAGMA 1.0 release using AMD's Accelerated Parallel Processing Math Libraries (APPML) for BLAS calculations.
- Performance for OpenCL can be worse than for CUDA, depending on available hardware (see graph below).

Basic operations are similar to the CUDA bindings (operator overloading will be added eventually):

- cl_fltarr(N) for cl_fltvar(N)
- cl_putvar(x) for cl_putvar(x)
- cl_getHandle() for cl_getHandle()
- clMAGMA 1.0 release using AMD's Accelerated Parallel Processing Math Libraries (APPML) for BLAS calculations.
- Performance for OpenCL can be worse than for CUDA, depending on available hardware.

Custom expressions can easily be created and evaluated in the IDL OpenCL bindings:

- cl_var(x)
- cl_var(y)
- cl_var(x, y, z, /nozero)
- kernel = cl_compile("z[i] = 2 * x[i] + sin(y[i])", $ x, y, z, /nozero)
- status = cl_execute(kernel, [ x: dx, y: dy, z: dz ])

The SIMPLE keyword in the CL_COMPILE call indicates that the program compiled is just the expression to evaluate for each element of the arrays, not the test of an entire kernel which is the default.

GPULib can use MAGMA to perform GPU accelerated LAPACK calculations. The IDL bindings for the low-level MAGMA routines are automatically generated, but GPULib also offers some higher-level routines mimicking the LA* routines present in the IDL library. For example, GPULIB INVERT is comparable to LA_INVERT, inverting a non-singular matrix:

\[ d\text{inverse} = \text{gpulib} \text{inver}(d\text{a}, \text{lhs}=d\text{inverse}) \]

While these routines document their technique for performing their operations, but do not require much user knowledge of the underlying mathematics. Calling the low-level routines directly in MAGMA requires more knowledge of the mathematical algorithms and a bit more cumbersome notation. For example, the following example shows how to call the MAGMA version of SGELS:

\[ \text{status} = \text{gpulib} \text{sgels}([\\text{byte}(\text{"N"})], \[ a, b, c, rwork, lwork, \]) \]

References

- MAGMA
- OpenCL
- NVIDIA.com/cuda
- CUDAvice.com/object/cuda_home_new.html
- GPULib www.txcorp.com/products/GPULib

Copyright © 2012 Tech-X Corporation

CUDA/OpenCL bindings vs. CPU IDL

Performance of the CUDA/OpenCL bindings reaches nearly 60x faster than standard IDL for some MAGMA computations. While the OpenCL bindings do not perform quite as well, reaching over 42x faster than standard IDL, they scale in a similar fashion.

Deconvolution of Hubble Space Telescope image done with GPULib. On a system with an NVIDIA Tesla C2070 and Intel(R) Xeon(R) CPU X5650 @ 2.67GHz, GPULib computed the deconvolution in 0.317 sec vs. 2.15 sec for the standard IDL array operations (about 6.8 times faster).