GPU Linear algebra extensions for GNU/Octave

L B Bosi[1], **M Mariotti**[2] **and A Santocchia**[3]

[1] Dipartimento di Fisica, Camerino, IT
[2] Dipartimento di Fisica, Perugia, IT
[3] INFN and Dipartimento di Fisica, Perugia, IT

E-mail: leonebosi@gmail.com, mirko.mariotti@fisica.unipg.it,
attilio.santocchia@pg.infn.it

Abstract. Octave is one of the most widely used open source tools for numerical analysis and liner algebra. Our project aims to improve Octave by introducing support for GPU computing in order to speed up some linear algebra operations. The core of our work is a C library that executes some BLAS operations concerning vector- vector, vector matrix and matrix-matrix functions on the GPU. OpenCL functions are used to program GPU kernels, which are bound within the GNU/octave framework. We report the project implementation design and some preliminary results about performance.

1. Introduction

GPU devices are impacting enormously the science computing, mainly due to their very interesting performances in terms of GFlops per Watt per Euro. Just to have an idea, looking at the top 500 list we discover that the first and the third ranked supercomputers in the word are GPU powered. The strong invasion of GPU hardware in the field of science computing has been stimulated and supported by a lot of important GPU software projects born all around the world.

The main purpose of our work is to make the computational power of modern GPUs accessible within octave, the popular free software scientific tool.

2. Octave

Octave provides functions to solve numerical solution of linear and nonlinear problems, performing other numerical experiments, plotting facilities. Octave can be user via command line, using a Matlab like language and also via program scripts for a non–interactive execution.

GNU Octave was written by John W. Eaton and many others and it is also freely redistributable software and can be redistribute and/or modified under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation.

Octave is a free software and for that the Octave community encourages to help making the software more useful. Contributions can be provided by writing additional functions for it, and by reporting any problem occurred.

3. OpenCL

OpenCL means Open Computing Language. This language has been developed to permit writing programs across multi-platform realized with CPU, GPU and other processors.
OpenCL was initially developed by Apple, which holds trademark rights, and refined into an initial proposal in collaboration with technical teams at AMD, IBM, Intel, ARM and Nvidia. Apple submitted this initial proposal to the Khronos Group. On June 16, 2008 the Khronos Compute Working Group was formed with representatives from CPU, GPU, embedded-processor, and software companies. This group worked for five months to finish the technical details of the specification for OpenCL 1.0 by November 18, 2008. This technical specification was reviewed by the Khronos members and approved for public release on December 8, 2008.

OpenCL is analogous to the open industry standards OpenGL and OpenAL, for 3D graphics and computer audio, respectively. OpenCL is managed by the non-profit technology consortium Khronos Group. The OpenCL include a language based on C99 for writing the functions that are executed on OpenCL devices. These functions are called kernels. Moreover the framework provides an API used to define and control the various platforms. OpenCL gives to the applications access to the graphics-processing unit for non-graphical computing, and a task-based or data-based parallel computing models.

We use OpenCL to develop the GPU linear algebra library extension for octave.

4. Extending octave

The aim of our work is to introduce the GPU computing inside the octave framework, bringing the OpenCL power within GNU Octave. This approach promises much better performance on big matrices/vectors size context, where GPU devices can express better their characteristic.

In particular we start analysing the feasibility of this integration, optimizing the GPU context handling and producing the first prototype version of the octave-gpu library. By design, this version provides simple vector-vector, vector-matrix, matrix-matrix operation such as fundamental operators and determinants. The work has been complicated due to the octave and OpenCL peculiarity, but also due to the poor octave documentation on object extensions, forcing as on analysing octave sources and pre-processed code. This work permitted to reconstruct enough advanced information on octave design, needed to introduce our optimized OpenCL support.

The standard way to extend Octave is using the so-called OCT files. An OCT file is essentially a loadable-shared library that the interpreter may load at runtime and use. Octave provides a utility to create OCT files from sources: mkoctfile.

Let's take a look at a trivial OCT file:

```
#include <octave/oct.h>
#include <stdio.h>
#include <stdlib.h>
DEFUN_DLD(testme, args, , "Test OCT file")
{
 int nargin = args.length ();
 return octave_value(nargin);
}
```

This example adds an octave function which purpose is counting and returns the number of its own arguments. The DEFUN_DLD macro is needed to create the interpreter entry point of a new octave function (testme is this example). The files has to be named the same as the function it declares, in this case testme.cc will be the source and testme.oct the OCT file. The octave interpreter expects to find a called external function in an OCT file named in the same way. If we would like to put two or more functions within the same source file we have to take care that every function has its own file (eventually creating symbolic links or copying a single OCT file to various files). Creating the OCT file is straightforward: "mkoctfile testme.cc", and to use it from an octave interpreter (opened in the same directory where the OCT file is):

```
octave:1> testme(3,4,5)
ans =  3
```

4.1. The first extension test “vector_add”

Our first test to put OpenCL code within Octave is to create a vector_add.cc that take two octave arrays, copies them to a GPU device, perform the sum and copy the result back to octave. All the OpenCL Context, device and kernel initialization is inside such function. This was a feasibility level zero test about the OpenCL-octave that demonstrated the possibility of the integration and provided precioused information for the next step of the library that will be reported on the next chapters.

4.1.1. “vector_add” benchmarking

We wrote a simple benchmark under Octave to the vector_add level zero code. The code loops several times over different vector length the “vector add” command, where the length is changed in order to test the code increasing computational complexity. The GPU is expected to best work under high computing density. Moreover we loop each length in order to have some statistics.

We develop more implementation of the benchmark, in order to have number to compare together:

- Octave native vector sum: this use the native octave “+”
- Vector_add: this is the first GPU implementation
- Vector_add_precompiled_kernel: this implements a precompiled kernel loading
- Vector_add_static: this is not called from the octave interpreter but is called as a function (we use it as gain upper limit)

The result are reported below, where we report execution time versus vector size. The hardware used for the test is a CPU Intel Xeon E5520 2.27GHz and a GPU Nvidia Tesla C1060.

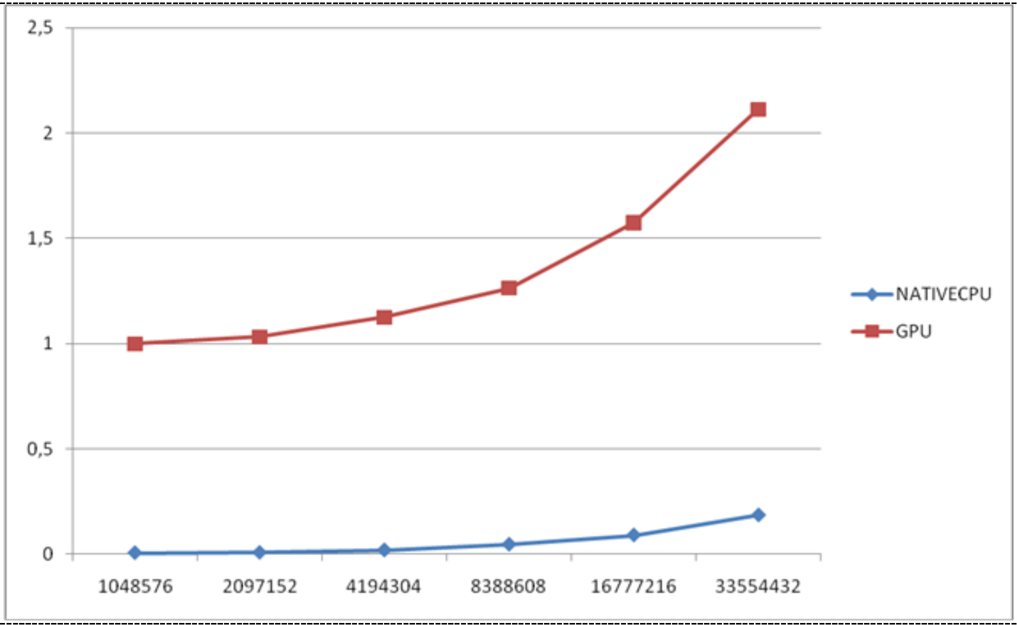

Figure 1. In this figure we show the comparison between GPU and CPU performance, reporting the "vector_add" average processing time (s) vs vector size.

With the zero level implementation we show the octave-GPU link feasibility and the implementation perspectives and first evident criticality, because the GPU performance are much worsted than the CPU. Some overheads were clear at design time and others were shown analysing the octave data/object handling. At the first order:

- ☐ The OpenCL initialization occurs every time an operation is performed.
- ☐ The OpenCL kernel compilation occurs every time an operation is performed.
- ☐ The Vector data are copied every time from host/host (double buffering) and from the GPU/host memory.

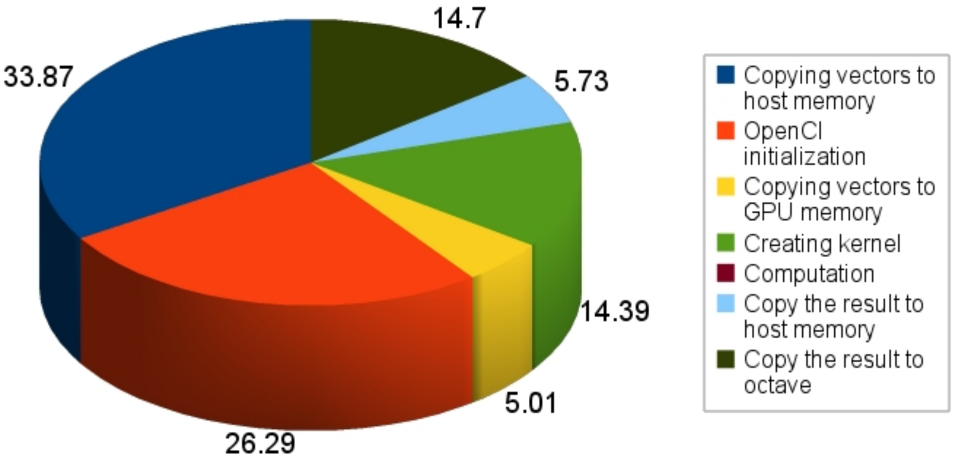

Figure 2. In this figure we show the "vector_add" profiling, reporting the percentage of time spent on the specific step.

The result analysis shows at first sight the main latency causes, mainly due to OpenCL initialization, and OpenCL /octave IO overheads.

We discovered that several overheads are added from OpenCL to the octave code.
Especially the copy operations in and from the octave native elements bring many delays in.

The analysis on such results bring us to the version one implementation, introducing important optimization actions:

- Definition of a permanent GPU object to host OpenCL status variables and status object.
- Kernels pre-compilation and bufferization
- OpenCL device and memory with once initialization approach

The our GPU-octave implementation consist of two main object, GPUdevice and GPUobject, where a GPUdevice is the GPU representation inside the octave space, while a GPUobject is a GPU data representation inside the octave space.

Moreover we already introduced at the epoch of this article the overloading of the most common operators: +,-,*

4.1.2. The GPUdevice octave object

The purpose of the GPUdevice is to store all the OpenCL initialization data for a given GPU device id. This code implementation supports the multi device support, where the device id is given as initialization parameter. A GPUdevice object, such as oGPU, can be declared using the following sintax:

```
oGPU=GPUdevice(0)
```

Where “0” mean the GPU device 0, in our case the C1060.

When a GPU device object is declared the code execute the following operations:

- Compile and load into memory all the defined OpenCL kernels
- Initialize the GPU device
- Initialize the GPU context

4.1.3. The GPUobject octave object

Once a GPUdevice has been declared, the user can allocate several GPUobjects starting by that. The GPUobject are the objects used by the octave user to perform the vector/matrix manipulation. The syntax introduced for this purpose is:

```
A=GPUobject(oGPU,[1:100000])
```

The command creates a GPUobject, embedding the OpenCL context and pre-initialized data through the GPUdevice - GPUobject initial linkage. Moreover with this command the GPU memory is allocated and a vector of 100000 elements is stored, containing number from 1 to 100000.

The user can declare and initialize several GPUobjects starting from the same GPUdevice and the purpose of the library is to provide the common octave syntax for the vectors operations.

For example the user can declare two objects:

```
A=GPUobject(oGPU,[1:100000])
B=GPUobject(oGPU,[10:100010])
```

And perform a sum, difference or element-by-element multiplication:

```
C=A+B
D=A-B
E=A*B
```

4.1.4. Benchmarking

We benchmark the sum and product functions of the new version of the library following the similar approach of the zero level version. In detail we estimate the gain factor between CPU and GPU timing on vector size increasing about:

- A+B (SUM)
- A*B (PRODUCT)
- A+B+C (SUM3)
- A+B*C (TRIAD)

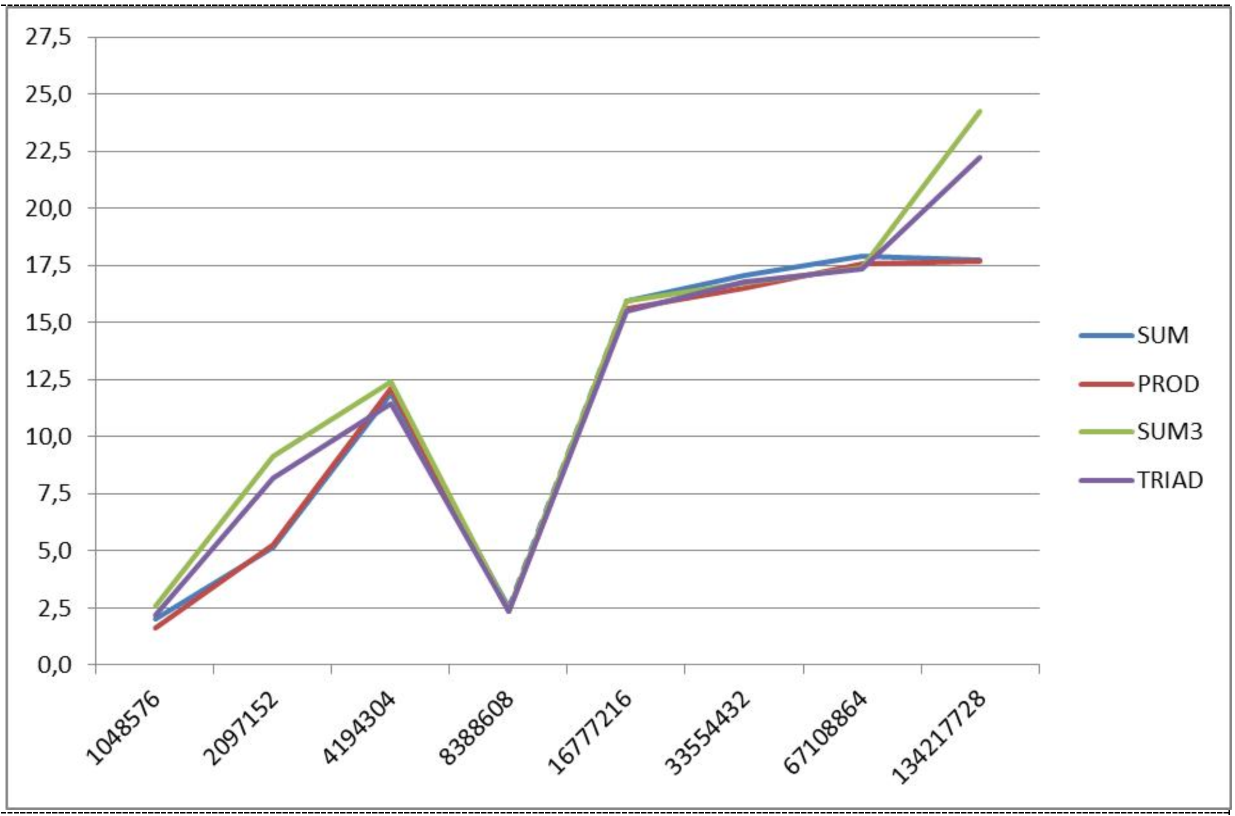

Figure 3. In this figure we report the processing gain factors of sum (a+b), mul (a*b), sum 3 terms (a+b+c), triad (a+b*c) operations between CPU and GPU on vector length change.

The first main result of this library version is the gain on data processing using GPU respect to the normal octave CPU implementation. This consideration is true in the range of the tests. We observe, as expected, a gain factor that roughly increase proportionally with processed vectors length. We observe a drop at 2^23 that needs investigation, probably due to the GPU computing blocks size that it hasn't been optimized at this step.

Sum and product report very similar performances with a gain between 2 and 17. The SUM3 and triad report a higher gain between 2.5 and 24. So it means that increasing the GPU computing volume the gain factor increases. This consideration lead us to deduce that the measured timing has still a systematic dominant delay component due to some overhead that can be again object of optimization.

5. Conclusions

Our work demonstrates the feasibility of GPU (OpenCL)-octave integration. Our library design allows getting a significant gain from the GPU implementation. In this version of the library we report gains between 2 and 24 respect to the normal CPU-octave computation. Now that the design responds positively, the work will continue finalizing the version 1 of the library implementing more operators and type to matrix.

References

[1] www.top500.org

[2] Strite S and Morkoc H 1992 *J. Vac. Sci. Technol.* B **10** 1237

[3] Jain S C, Willander M, Narayan J and van Overstraeten R 2000 *J. Appl. Phys.* **87** 965

[4] Nakamura S, Senoh M, Nagahama S, Iwase N, Yamada T, Matsushita T, Kiyoku H and Sugimoto Y 1996 *Japan. J. Appl. Phys.* **35** L74

[5] Akasaki I, Sota S, Sakai H, Tanaka T, Koike M and Amano H 1996 *Electron. Lett.* **32** 1105

[6] O'Leary S K, Foutz B E, Shur M S, Bhapkar U V and Eastman L F 1998 *J. Appl. Phys.* **83** 826

[7] Qian Z G, Shen W Z, Ogawa H and Guo Q X 2002 *J. Appl. Phys.* **92** 3683

[8] Guo Q X, Okada A, Kidera H, Tanaka T, Nishio M and Ogawa H 2002 *J. Cryst. Growth* **237–239** 1032

[9] Aderhold J, Davydov V Yu, Fedler F, Klausing H, Mistele D, Rotter T, Semchinova O, Stemmer J and Graul J 2001 *J. Cryst. Growth* **222** 701

[10] Mamutin V, Veskin V, Davydov V, Ratnikov V, Shubina T, Inanov S, Kopev P, Karlsteen M, Soderwall U and Willander M 1999 *Phys. Status Solidi* **176** 247

[11] Jenkins D W and Dow J D 1989 *Phys. Rev.* B **39** 3317

[12] Tansley T L and Egan R J 1992 *Phys. Rev.* B **45** 10942

[13] Wessel R, Koch C and Gabbiani F 1996 Coding of time-varying electric field amplitude modulations in a wave-type electric fish *J. Neurophysiol.* **75** 2280–93