COBOSLAB

Cognitive Bodyspaces: Learning and Behavior

Technical Report No. CoboslabY2010N003

30th of November 2010

CUDA implementation of $V1$ based on Gabor filters

Kevin L. Reif & Martin V. Butz

COBOSLAB, Department of Psychology

University of Würzburg

Röntgenring 11

97070 Würzburg, Germany

http://www.coboslab.psychologie.uni-wuerzburg.de

CUDA implementation of $V1$ based on Gabor filters

Kevin L. Reif* Martin V. Butz†

Abstract

This technical report describes our parallel implementation of various visual filters. The focus lies on the implementation of Serre et al. (2007)'s Gabor filter-based layer $S1$ and the applied max operator in layer $C1$ – both of which mimic activities found in the primary visual cortex $V1$. The implementation runs on CUDA-capable graphics cards and provides at least one order-of-magnitude speed-up compared to a standard serial implementation.

1 Overview

1.1 Main Capabilities

The program allows the flexible filtering of any given squared gray-scale image using a battery of Gabor filters. Also several other simple image filters are supported. Figure 1 gives a simple example of a Gabor-filtered image. All filters are implemented in a parallel way, running on a CUDA-capable GPU (NVIDIA, 2010b).

The basic program first loads a specified image, converts it, and hands it over to a CUDA-capable graphics card. On the GPU, a parallel implementation of the standard Gabor filter is applied to the image. In one step, 16 sizes of the filter (ranging from 7x7 to 37x37) are used to produce 16 filtered images. This is done for 4 different rotational angles (0°, 45°, 90°, and 135°) resulting in a total of 64 filtered images. These are then written back to the hard disk in JPEG format. The parameters used in the creation of the Gabor filter matrices such as scaling factors, aspect ratios, and wavelengths are taken from Serre et al. (2007)

1.2 Additional Capabilities

In addition to the Gabor filter, our program is also able to perform filtering operations using a Gauss filter, a Sobel filter, a Laplace filter. Also the $C1$ layer implementation of Serre et al. (2007) is supported optionally, which

*kevin.reif@stud-mail.uni-wuerzburg.de
†butz@psychologie.uni-wuerzburg.de

Figure 1: Unfiltered and Gabor filtered image.

performs local MAX operation on the produced 64 filtered images. In this case, only the images of the $C1$ layer are handed back to the CPU.

2 General Structure

Our program is divided into different parts each with their specific tasks. It is written in Java and C for CUDA with Java's Native Interface (JNI) and a proxy C file forming the link between the two different architectures.

2.1 Java

The program's command line interface and all image manipulation capabilities reside in the Java code. This includes loading from and writing the images to the hard disc, converting the images for further manipulation on the GPU, and performing necessary cropping and gray-scale conversion.

2.2 JNI

The Java Native Interface comes into play when using functionality that is written in the C for CUDA programming language. JNI provides the necessary header files for a proxy file written in C that implements the native methods from the Java code.

2.3 C Proxy

The part of the program written in plain C would under normal circumstances implement the native methods directly. However, since we make use

of a CUDA-capable GPU, the C file's task is to call the filter implementations written in C for CUDA. It only acts as a proxy for a more convenient usage.

2.4 CUDA

All image manipulations based on convolutional transformations are performed on the GPU. For that purpose, this part of the program is written in C for CUDA, which is an extension to the C programming language with few restrictions on C (NVIDIA, 2010b).

3 Required Hardware

3.1 General Hardware

In order to successfully compile and run the presented architecture we recommend a system that meets the following specifications:

- 2+Ghz CPU or better
- 1+GB RAM or better
- CUDA-capable GPU

3.2 CUDA Specific Hardware

CUDA is a proprietary architecture provided by the NVIDIA company. Therefore, only NVIDIA GPUs are capable of running CUDA code. A full list of CUDA-enabled devices and their respective compute capabilities can be found online (NVIDIA, 2010a).

4 Required Software

4.1 Operating System

We developed our architecture on an Ubuntu 10.04 64bit Linux operating system. All pre-compiled versions of our program are intended for this environment. Ubuntu version 10.10 has also been successfully tested.

It is possible to compile a 32bit version of the program on an Ubuntu Linux system with the provided makefile. Other Linux distributions have not been tested. For further instructions on how to compile the sources under Linux please refer to section 6.

Windows users also will have to compile their own version of the source code. This has not been tested, however. For further comments on compiling under Windows, please refer to section 6.3.

4.2 CUDA Specific Software

It is necessary to install the CUDA development package provided by NVIDIA. The package consists of the appropriate CUDA-capable drivers for your NVIDIA graphics card, the CUDA runtime environment, and the CUDA SDK. They are all available on NVIDIA's CUDA developer web site (NVIDIA, 2010b). Instructions on how to set up the CUDA development environment and links to the respective packages can be found in section 5.

4.3 Additional Software

If you plan on changing or adding source code to the existing one, the following additional software might be useful:

- As Java IDE we used the Eclipse SDK Version 3.5.2.
- Instead of the java-6-openjdk shipped with Ubuntu, we prefer SUN's version java-6-sun-1.6.0.22 due to its additional documentation.
- For C or C for CUDA code, respectively, we used the CDT plugin for Eclipse.

For further notes on how to set up Eclipse, the Java JDK, and CDT please see Sec.5.

5 Setup

If not otherwise stated, all instructions refer to a Ubuntu 10.04 64bit Linux as underlying operating system.

5.1 CUDA environment

For further documentation on installing the CUDA environment on other operating systems please refer to the "Getting Started Guide" provided by NVIDIA. All necessary software and documentation for the CUDA toolkit can be found on the NVIDIA developer zone (NVIDIA, 2010c).

5.1.1 Drivers

First download the appropriate driver for your NVIDIA GPU. If you prefer the non-developer drivers, they are available under
http://www.nvidia.com/Download/index.aspx?lang=en-us.
Select your model and operating system (64bit Linux). Current graphics cards should work with driver version 190.x or higher. We recommend version 253.x or 260.x. Please make sure that you install the drivers from the NVIDIA web site, as driver versions that ship with Ubuntu (including

restricted and multiverse sources) might not work.

The following steps might be necessary to ensure a successful installation:

1) Blacklist the following modules by typing
`sudo nano /etc/modprobe.d/blacklist.conf` into your terminal window. Add the following lines to the config file, add a final space after the last entry and safe:
```
blacklist vga16fb
blacklist nouveau
blacklist rivafb
blacklist nvidiafb
blacklist rivatv
```

2) Uninstall your current NVIDIA driver (if you should have a version installed that shipped with Ubuntu) by typing
`sudo apt-get --purge remove nvidia-*` into your terminal.

3) Restart Ubuntu. It should start in low graphics mode. Select "Exit to terminal".

4) Remove nouveau by typing
`sudo apt-get --purge remove xserver-xorg-video-nouveau`

5) Repeat step 3.

6) Make the following final adjustments:
`sudo nano /etc/modprobe.d/blacklist-framebuffer.conf`
Comment out: `blacklist vesafb`
Add: `blacklist vgafb16` and safe.
Type `sudo nano /etc/initramfs-tools/modules`
Add: `fbcon` and `vesafb` (don't forget the last space) and safe.
Finally update initrams by typing
`sudo update-initramfs -u`

7) Repeat step 3.

8) Install the appropriate drivers by typing
`sudo sh NVIDIA-Linux-x86_64-XXX.XX.run`
where XXX.XX is your version.

5.1.2 CUDA Toolkit

Download the CUDA toolkit and install it by typing
`sudo sh cudatoolkit_X.X_linux_64_ubuntuX.XX.run` into your terminal. It is important to install the runtime environment as root by using `sudo`. The default directory provided by the run-file is recommended.

In your home folder add the following line to the config file `.bashrc`:

```
export PATH="/usr/local/cuda/bin:$PATH"
export LD_LIBRARY_PATH="/usr/local/cuda/lib64:
/usr/local/cuda/lib:$LD_LIBRARY_PATH"
export CPLUS_INCLUDE_PATH="/usr/local/cuda/include"
export LIBRARY_PATH="/usr/lib/nvidia-current"
```

5.1.3 CUDA SDK

In order to check if driver and runtime environment have been successfully installed, download the CUDA SDK from (NVIDIA, 2010c). Install it by typing
`sh gpucomputingsdk_3.1_linux.run` into your terminal. Please note that in this case the installation should be done by a regular user not root. Again the default directory is recommended. After the installation is complete, adjust the `common.mk` file found in
`/home/[user]/NVIDIA_GPU_Computing_SDK`
`/C/common`. Find the first instance of `NVCCFLAGS` and change it to:
`NVCCFLAGS := --compiler-options -fpermissive`. Then compile the example files by navigating to
`/home/[user]/NVIDIA_GPU_Computing_SDK/C/` and typing `make` into your terminal window. You should now be able to run the executable files found under
`/home/[user]/NVIDIA_GPU_Computing_SDK`
`/C/bin/linux/release`. Type `./deviceQuery` to check if your graphics card is capable of running the CUDA code.

5.2 Eclipse

5.2.1 Eclipse SDK

To install the Eclipse SDK under Ubuntu, type `sudo apt-get intall eclipse` into your terminal window. This will install the Galileo version of eclipse under Ubuntu 10.04. The 64bit and 32bit version of Ubuntu do not differ here.

5.2.2 CDT

The CDT plugin for C and C++ development can be installed by selecting "Help", then "Install New Software..." from the Eclipse menu and adding `http://download.eclipse.org/tools/cdt/releases/galileo` as software location. Please make sure that you are in fact using the Galileo version of Eclipse. Otherwise refer to the CDT homepage for alternate versions.

To enable text-highlighting for C for CUDA source code choose Window, Preferences from the Eclipse menu. Select C/C++, File Types and add *.cu as C++ source file.

5.3 SUN Java JDK

In order to install SUN's version of the Java 6 JDK it might be necessary to add the appropriate repository to your software sources list by typing `add-apt-repository "deb http://archive.canonical.com/ lucid partner"` into your terminal window. After that you can install the JDK by typing `sudo apt-get install sun-java6-jdk`. This works for both the 64bit and 32bit version of Ubuntu.

6 Compilation

If you are using the 64bit version of Ubuntu 10.04 or 10.10 you can use our pre-compiled version of the program.

To allow you to compile your own version of the CUDA code we provided a makefile. Please make sure to adapt the following lines in the makefile to match your local configuration:
`CUDA_INSTALL_PATH := /usr/local/cuda`
`CUDA_SDK_PATH := /home/[user]/NVIDIA_GPU_Computing_SDK`
`JDK_PATH := /usr/lib64/jvm/java-6-openjdk` or
`JDK_PATH := /usr/lib64/jvm/java-6-sun-1.6.0.22` or appropriate version.

6.1 64bit Linux

6.1.1 Java Sources

All Java sources (`.java`) have to reside in the same directory as the makefile. The makefile specifies the appropriate parameters for the javac compiler and the javah header creator.

To compile the Java sources navigate to the directory that holds the source files and type `make java`. This will remove the old .class files and the `filters.so` dynamic library should it exist. It will then call the javac compiler and javah header creator. The resulting header (`Filters.h`) has

to be included in the `proxy.c` file. After that, all CUDA files have to be compiled again to create a new `filters.so` dynamic library.

Should you only want to compile a changed Java file without having to recompile the CUDA files, type `make javaonly`.

6.1.2 CUDA Sources

All CUDA sources (`.cu`) have to reside in the same directory as the makefile. The makefile specifies the appropriate compiler parameters. All `.cu` files in the directory and the `proxy.c` file are compiled into object files (`.o`) and then linked together into one dynamic library (`filters.so`). This is done automatically by typing `make` into your terminal window. The `filters.so` is loaded in the loader method inside the `Filters.java` class.

Should you want to change the file name of the dynamic library, please adjust the makefile at the following line: `BIN := filters`.

6.2 32bit Linux

It is possible to use the provided makefile to compile the sources for a 32bit Linux distribution. For that purpose change the compiler flags `-m64` to `-m32` and any references from `lib64` to `lib`. The provided `libcutil_x86_64.so` dynamic library can still be used.

6.3 Microsoft Windows

In order to use the CUDA functionality under Microsoft Windows, it is necessary to compile the CUDA sources into a `.dll` dynamic library. Please refer to the CUDA User Guides and the documentation of your C compiler for more information.

7 Usage

7.1 Available Filters

As mentioned in Sec.1 the main feature of the program is the parallel Gabor filter running on a CUDA GPU. It takes as input a square gray-scale JPEG image with a side length that is a multiple of 16. The outputs are 64 images of the same size, one for each combination of filter matrix size and rotational angle.

In addition to the standard Gabor filter, it is also possible to choose the S1 extension, in which case a MAX pooling function is used to create 32 output images, which corresponds to the $C1$ units described in Serre et al. (2007). Those images are the result of locating the maximum value of a filter scale band consisting each of 2 Gabor filter outputs of consecutive filter sizes.

Furthermore, there are a Sobel edge detection filter, a Laplace edge detection filter, and a Gauss blurring filter available. To convert images into the appropriate square shape and a side length that is a multiple of 16 there is a `convert` option that can be used with the filter parameter. It takes any JPEG image as input and returns a square gray-scale JPEG image that can be used as the input for the other filters mentioned above.

7.2 Command Line Interface

To run the program navigate to the directory of the Java `.class` files and type `java Filters [options]`.

The following command line options are available:

- `-f [gauss|sobel|laplace|gabor|gaborMax|convert]` filter selection - required
- `-s [1|2|4|8|16]` filter size for Gauss filter - optional, default is 4
- `-i [filename]` file name of the input image in JPEG format - required
- `-o [filename]` file name of the output image - required
- `-d [input directory]` path to input image - optional ("resources" in the `.class` files' directory is set as default)
- `-t [target direcoty]` path to output image - optional, if omitted "resources" in the `.class` files' directory is used
- `-m [gpu|cpu]` flag for selecting GPU or CPU for computations - optional, default is GPU
- `-w [t|f]` flag for selecting if output files are to be written to disk - optional, default is set to true

8 Concluding Remarks

This code is distributed for academic purposes with absolutely no warranty of any kind, either expressed or implied. We are not responsible for any damage from its proper or improper use. Saying this, we want to however emphasize that we did our best to provide a useful CUDA version of the implemented visual filters.

Even if the software will be refactored and some aspects will not be included in further releases, feel free to use, modify and distribute the code with an appropriate acknowledgment of the source. In all potentially resulting publications, please include a citation of this technical report.

If you have further questions or suggestions, please do not hesitate to contact the authors. Also, please report any bugs or other inconsistencies you may found in the source code to the authors.

References

NVIDIA (2010a). CUDA GPUs. http://www.nvidia.com/object/cuda_gpus.html.

NVIDIA (2010b). CUDA zone. http://www.nvidia.com/object/cuda_home_new.html.

NVIDIA (2010c). NVIDIA developer zone. http://developer.nvidia.com/page/home.html.

Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., and Poggio, T. (2007). Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:411–426.