

GPUgaussMLEv2 Solution in Visual Studio 2010

using Windows 7, Matlab 2010b and CUDA 4.0

Ronald Ligteringen

June 16, 2011

1 Introduction

This manual explains how to setup your computer to use the GPUgaussMLEv2 Solution in Visual Studio 2010. It is not a manual on how to use the GPUgaussMLEv2 software itself.

The GPUgaussMLEv2 program is available as a C library and a Matlab library. Both libraries can be created using Visual Studio 2010. For this a Solution has been created separating the two libraries from the core GPU code. In total five Projects are defined:

GPUgaussMLEv2 This is the CUDA core code. All interface independent code goes here.

cpp_GPUgaussMLEv2 This is the C interface for the CUDA core.

mex_GPUgaussMLEv2 This is the mex interface for the CUDA core.

test_cpp_GPUgaussMLEv2 This program can be used to test the C interface or as an example.

test_mex_GPUgaussMLEv2 This m-script can only be run from inside Matlab. It is added to the Solution to provide a complete package.

2 Prerequisites

To build and test the libraries you will need a PC with a CUDA capable graphic card. Tests were done on a Dell Optiplex GX620 with the NVIDIA Quadro FX 1700. As you will be switching from Visual Studio to Matlab and back you need sufficient memory in the order of 4GB. Further software prerequisites are given below:

Windows This Solution was tested on Windows 7 64-bit. Presumably you could also use 32-bit although I am not sure if this would allow you to create 64-bit libraries. Most likely XP and Vista will also work.

Visual Studio 2010 This is a VS 2010 Solution and will not work on older versions of Visual Studio. Make sure you install the 64-bit compiler if you need to make 64-bit libraries.

Matlab This Solution was tested with Matlab 2010b 32-bit **and** 64-bit. It is likely that older versions will also work. Although Matlab is known to change a lot of features with their upgrades.

CUDA This Solutions was tested with CUDA 4.0. This version contains both 32- and 64-bit.

3 Software Installation

The preferred order of installation is: 1. Visual Studio, 2. Matlab and 3. CUDA. You need to tell Matlab which compiler to use by giving the following command in Matlab: `>> mex -setup`. Then select the Visual C++ compiler. If you are going to create both 32- and 64-bit libraries you will have to do this for both versions of Matlab.

NOTE: only one settings file (`mexopts.bat` is used in Matlab for both 32- and 64-bit but with different settings; therefore you have to rename the settings file after creation with `>>mex -setup` for later use: `mexopts32.bat` and `mexopts64.bat`, these files can be found here: `...\Application Data\MathWorks\MATLAB\R2010b` (see section 4.3).

With the installation of CUDA scripts are provided to include support in Visual Studio. If these scripts are not automatically run during installation you can find them here:

`C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v4.0\extras\visual_studio_integration\`.

4 Settings in Visual Studio

The Solution contains five Projects with the core software (`GPUgaussMLEv2`), two interfaces (`cpp_GPUgaussMLEv2` and `mex_GPUgaussMLEv2`) and two tests (`test_cpp_GPUgaussMLEv2` and `test_mex_GPUgaussMLEv2`). The two interfaces build the final libraries and the tests can be used as examples or for testing the core and interfaces. Note that the mex-test has been added to the Solution to create a complete package but can only be run from Matlab! The appropriate settings in each Project are discussed below.

4.1 GPUgaussMLEv2

This is the core CUDA program of the libraries. Because older versions of GPU architecture (with `sm < 1.2`) do not allow for external CUDA kernel functions all CUDA code must be included in the interface code. This is done in the interfaces with `#include "GPUgaussMLEv2.cu"` and in the main core with `#include "MatInvLib.cu"` and `#include "GPUgaussLib.cu"`. The header file `GPUgaussMLEv2.h` contains the definitions used in the

core, interfaces and tests.

If you need to add a new project with CUDA code do the following:

1. Right click on the Solution and select `Add→New Project...`
2. Select `Win32 Console Application` and enter name
3. Select `Static library` and deselect everything else
4. Select `Build Customization...` (right click Project) and select `CUDA 4.0`
5. Create source file with `Add→New Item...`
6. Select `C++ File (.cpp)` and enter name
7. Select `Properties` of file and select `General→Item Type→CUDA C/C++`
8. Again select `Properties` and change `CUDA C/C++→Target Machine Platform` according to `Platform` defined on top of window
9. Rename file from `.cpp` to `.cu`

4.2 `cpp_GPUGaussMLEv2`

This is the C interface to the CUDA core. The Project is setup to build a DLL with one callable function `gpugaussmle` with the following parameters:

float *data input data, pointer to matrix

int sz size of array; `x=y` (array must be square)

int Nfitraw number of arrays

int PSFSigma point spread function sigma

int iterations number of iterations

int fittype which type of function (1=MLEfit, 2=MLEfit_sigma, 3=MLEfit_z, 4=MLEfit_sigmaxy)

float *d_Parameters output parameters, [*size = number_of_arrays*number_of_parameters*].

The number of parameters depend on the fittype (1=NV_P, 2&3=NV_PS, 4=NV_PS2)

float *d_CRLBs output CRLBs, [*size = number_of_arrays * number_of_parameters*]

float *d_LogLikelihood output loglikelihood, [*size = number_of_arrays*]

If fittype is 3 then the following parameters also must be given after the `*d_logLikelihood`:

float Ax, Ay, Bx, By, gamma, d, PSFSigma_y parameters needed for fittype 3 (MLEfit_z)

These extra parameters are handled through the varargs-method in `<stdarg.h>`.

4.3 mex_GPUgaussMLEv2

This is the mex interface to the CUDA core. This Project builds a mex library which can be called from within Matlab:

```
>> [d_Parameters, d_CRLBs, d_LogLikelihood] = mex_GPUgaussMLEv2(data,
    PSFSigma, iterations, fittype) % for fittype 1,2 and 4

>> [d_Parameters, d_CRLBs, d_LogLikelihood] = mex_GPUgaussMLEv2(data,
    PSFSigma, iterations, fittype,
    Ax, Ay, Bx, By, gamma, d, PSFSigma_y) % for fittype 3
```

The parameters have the same meaning as described in the `cpp_GPUgaussMLEv2` Project (see section 4.2).

The object files must be linked with the mex Matlab program. This can be set in the Properties of the Project:

1. Right click on the Project and select Properties
2. Select Configuration Properties→Build Events→Post-Build Event→Command Line
3. Enter the following line:

```
for 32-bit:
"C:\Program Files (x86)\MATLAB\R2010b\bin\mex"
-f "H:\Application Data\MathWorks\MATLAB\R2010b\mexopts32.bat"
-L"$(CudaToolkitLibDir)" -lcudart -outdir $(TargetDir) $(TargetPath)
```

```
for 64-bit:
"C:\Program Files\MATLAB\R2010b\bin\mex"
-f "H:\Application Data\MathWorks\MATLAB\R2010b\mexopts64.bat"
-L"$(CudaToolkitLibDir)" -lcudart -outdir $(TargetDir) $(TargetPath)
```

NOTE: the H: being the home-directory containing the two `mexopts` files and also note the space after `-f` and `-outdir` and lack thereof after `-L` and `-l`!

4.4 test_cpp_GPUgaussMLEv2

This is the test file for the C interface. The program reads an input file, runs the appropriate kernel and returns an output file. The input file can be generated with the mex test program in Matlab (see section 4.5). The output file can be read and compared with the mex interface output using the same mex test program. Make sure to supply the right parameters used for creating the input data. Also note that the pathname of the input and output data must be modified. To simplify the data handling only the `d_Parameters` are stored.

4.5 test_mex_GPUgaussMLEv2

The file `run20.m` is the test script for the mex interface. It can only be used from within Matlab and it needs the m-script `finiteGaussPSFperf.m` for generation of the input data. The script also allows you to create the input file for the C interface test and compare the result of the C interface with the mex interface. To do this follow the instructions in the comments in the script.