Quasar: installation guide

Bart Goossens

October 30, 2023

Contents

1 Prerequisites

- GPU - to take advantage of the GPU capabilities of Quasar, it is required to have a (recent) NVidia GPU with compute architecture of at least 1.3. See https://developer.nvidia.com/cuda-gpus to check whether your device is supported. In case no supported GPU is present, the Quasar programs can only be compiled and run using the CPU computation engine.
- GPU - NVidia CUDA 4.2 (or higher) is required. Recommended is to install CUDA 7.5. Download the CUDA package from here: https://developer.nvidia.com/cuda-downloads (or use the google search terms "download cuda").
- Runtime - for its special dynamic code generation abilities and to be platform-independent, Quasar requires the .Net Framework 4.0 (or higher) to be installed. The latest version is 4.6. On Windows PC's the .Net Framework is shipped with Windows, so nothing needs to be done. On Linux/MAC, MONO (v2.0 or later) is required (see http://www.mono-project.com).
- Quasar Redshift (IDE for Quasar) requires GTK 2.24 to be installed. Under windows, you need to use a 64-bit version of GtkSharp (send an email to quasar@telin.ugent.be, currently this version is not available online).
- *Warning*: do not install GTK 3.xx, because GTK# is not compatible with this version of GTK.

2 Windows

Update: there is a 64-bit windows installer available. We recommend using the Windows installer, rather than the manual instruction steps below.

1. Make sure whether you are running a 32-bit version of Windows, or a 64-bit version of Windows. Check if you installed the 32-bit driver version of CUDA, or the 64-bit driver version.
2. If necessary, copy the files from the directories `OS_Runtimes\32bit_Windows` to the directory of Quasar.exe. Note: on some computers this step is not needed - the Quasar runtime automatically detects whether it is run in 32-bit mode or 64-bit mode and selects the appriopriate binary directory.
3. In the root directory of the installation package is now the 32-bit version of Quasar.exe (this version is up to 2 times faster than the 64-bit version); in case you would like to run the 64-bit version of Quasar, you can find this executable in the directory OS_Runtimes\64bit_Windows.
4. Install a C/C++ compiler with OpenMP support. Currently, the following compilers are supported:
   - Intel C/C++ v.11
   - Microsoft Visual C/C++ v.9.0 (VS 2008)
   - Microsoft Visual C/C++ v.10.0 (VS 2010)
   - Microsoft Visual C/C++ v.11.0 (VS 2012)
   - Microsoft Visual C/C++ v.12.0 (VS 2013)
   - MingW GCC
   - Clang 2.5 or newer

   The compiler is selected according to the above order. For example, if both Visual C++ as Intel C++ are available, the Intel compiler will be used.

   If your favorite compiler is not listed here, don't panic. In that case, modify the batch file `cc_script.bat` to include your compiler (and send a copy of the modified `cc_script.bat` to `quasar@telin.ugent.be`). Make sure that you include entries for both 32-/64-bit.
5. Optionally, Quasar can be natively pre-compiled, using the `ngen` (Native Image Generator) tool. This may increase the compilation speed of Quasar programs by 30-40%. Run the following command for your command prompt:

   ```
   c:\Windows\Microsoft.NET\Framework\v4.0.30319\ngen.exe install Quasar.exe
   ```

3 Linux

3.1 Supported platforms & distributions

Many platforms are supported, but not all of them have been tested. Extensively tested platforms include x86 and x86-64; although Quasar also works on ARM (note: for ARM with hardware floating point support, a recent version of MONO is required). Other platforms that MONO supports are SPARC, PowerPC, IA64 Itanium2 and MIPS. Because the back-end code generator relies on `gcc` and because the compiler options are more or less the same for each platform, Quasar should work relatively flawlessly on all of these platforms. If not, you may post a bug report.

Quasar should run well on all major linux distributions. The distributions on which we test the most are Ubuntu, Xubuntu and Linux Mint.

3.2 Compiler support

The linux version of Quasar currently only supports GCC and Clang for native C/C++ compilation. However, other compilers can easily be added by modifying the `cc_script.sh`. If you do so, send the updated `cc_script.sh` to `quasar@telin.ugent.be`.

3.3 Prerequisites

The following software needs to be installed first:

- GCC (should be installed on most linux distributions by default)
- GTK 2.xx (normally installed by default, e.g., package `libgtk2.0-0`)
- MONO (look for the package `mono-complete` in the synaptic package manager)
- GTK# (look for the package `gtk-sharp2` in the synaptic package manager)

3.4 Automatic installation

Dirk made a linux installation script in the main Quasar folder with name "`install.sh`". Simply run the script from the terminal (`./install.sh`), Quasar should be installed in `/usr/local` or `{your_home}/local`. Furthermore, a taskbar icon is created for Quasar.

3.5 Manual installation steps

Important: some changes to `~/.bashrc` and `~/.bash_profile` are required, this is to get CUDA running:

1. add the following text add the bottom of `~/.bashrc`

```
source /etc/profile
. /usr/local/cuda/profile.sh
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

2. put the following text in `~/.bash_profile`

```
source ~/.bashrc
```

3. test whether nvcc runs correctly:

   - Open a new terminal
   - Enter "`nvcc --version`"
   - If correct, the output should be as follows:

```
Copyright (c) 2005-2010 NVIDIA Corporation
Built on Mon_Jun__7_18:56:31_PDT_2010
Cuda compilation tools, release 3.1, V0.2.1221
```

4. Adjust file permissions of a couple of files (important!)

```
cd ~/Quasar  [or your own Quasar directory]

chmod +x *cc_script.sh && chmod +x Quasar.exe
```

   Note: the permissions of the `*cc_script.sh` files are updated automatically by Quasar, if necessary.

5. Run and check if quasar is working properly

```
./Quasar.exe -version
```

   The output should be as follows:

```
-------------------------------------------------------------------------------
Quasar 1.0.0.0 - build date 8/13/2012 3:32:29 PM - 64-bit
Copyright © Bart Goossens 2011-2012
-------------------------------------------------------------------------------
OS: Unix 2.6.32.30
CPU - unknown processor [4 cores]
CUDA (driver version: 3.1, runtime version: 3.1, 32-bit)
GPU - GeForce GTX 460 [Mem: 1023 Mb, Compute cap.: 2.1]
Clock rate: 1430000 Hz
Can map host memory: 1
Supports concurrent kernels: 1
Integrated with CPU memory: 0
```

6. Check the output of the previous step if CUDA is listed. If there is no mentioning of CUDA, then the CUDA runtime is not found. Then use the test tools to check if CUDA is installed:

   - Enter `cd Tools; make` to recompile the test utilities.
   - 32-bit linux: `chmod +x Tools/test_cuda32 && ./test_cuda32`
   - 64-bit linux: `chmod +x Tools/test_cuda64 && ./test_cuda64`

7. If CUDA is working correctly, the output should be as follows:

```
==============================
CUDA Testing program
July 2012, Bart Goossens.
==============================
Number of CUDA devices: 1

CUDA Device #0
Major revision number:          2
Minor revision number:          1
Name:                           GeForce GTX 460
Total global memory:            1072889856
Total shared memory per block:  49152
Total registers per block:      32768
Warp size:                      32
Maximum memory pitch:           2147483647
Maximum threads per block:      1024
Maximum dimension 0 of block:   1024
Maximum dimension 1 of block:   1024
Maximum dimension 2 of block:   64
Maximum dimension 0 of grid:    65535
Maximum dimension 1 of grid:    65535
Maximum dimension 2 of grid:    1
Clock rate:                     1430000
```

```
Total constant memory:          65536
Texture alignment:              512
Concurrent copy and execution:  Yes
Number of multiprocessors:      7
Kernel execution timeout:       Yes

Size of host pointer: 8 bytes
Size of device pointer: 4 bytes
Size of cufftHandle: 4 bytes
Size of CUmodule: 8 bytes
Size of CUfunction: 8 bytes
Testing cuFFT...
cuFFT mean abs error: 1.91483e-05a
```

8. TROUBLESHOOTING: if cuda is not working, check if the LD_LIBRARY_PATH is set correctly:

   ```
   echo $LD_LIBRARY_PATH
   ```

   The output should contain `/usr/local/cuda/lib64` (64-bit linux) or `/usr/local/cuda/lib` (32-bit linux). Then check if the CUDA runtime libraries are installed correctly:

   ```
   locate cudart.so
   locate cufft.so
   ```

   If CUDA is installed correctly, these libraries should be in the same folder as the `LD_LIBRARY_PATH`.

9. Look at the output of `test_cuda32` or `test_cuda64`: Check the line "Size of device pointer: 4 bytes". If the size of the device pointer is 4 bytes, then the GPU needs to be run in 32-bit mode. Therefore: execute the following command

   ```
   cp GPU_Runtimes/32bit_GPU/CUDA.NET.dll .
   ```

   On the other hand, if the size of the device pointer is 8 bytes, then 64-bit GPU mode is required.

   ```
   cp GPU_Runtimes/64bit_GPU/CUDA.NET.dll .
   ```

10. Only if you are not running `Quasar.exe` from the folder in which Quasar is installed (e.g. using external gedit plugins): adjust `Quasar.config.xml`, and set NVCC_PATH and CC_PATH to the full directory of Quasar, e.g.

    ```
    <setting name="NVCC_PATH">
        <value>~/Quasar/nvcc_script.bat</value>
    </setting>
    <setting name="CC_PATH">
        <value>~/Quasar/cc_script.bat</value>
    </setting>
    ```

    Note that Quasar will automatically select the correct extension (.bat for windows, .sh for linux/unix, .osx.sh for MAC OS).

4 Mac OS X

Tested on: Mac OS X 10.6.8 and 10.11 (thanks to Bert Vandeghinste and Hiep Luong!)
What is needed as extra, and is not installed by default:

- Mono JIT compiler, downloadable from www.mono-project.com (tested with version 2.10.9)
- GTKSharp for MAC OS X
- GCC (g++) version 4.5.0 (or higher).

Installation steps:

- To install the prerequisites, the easiest option is to install Xamarin Studio (http://xamarin.com/download).
- Create the shells scripts `./quasar` and `./redshift` in the folder Quasar/NewestVersion:

```
# ./quasar:

#!/bin/sh
export DYLD_FALLBACK_LIBRARY_PATH="/Library/Frameworks/Mono.framework/Versions/Current/li
exec /Library/Frameworks/Mono.framework/Versions/Current/bin/mono Quasar.exe $@

# ./redshift

#!/bin/sh
export DYLD_FALLBACK_LIBRARY_PATH="/Library/Frameworks/Mono.framework/Versions/Current/li
exec /Library/Frameworks/Mono.framework/Versions/Current/bin/mono Quasar.Redshift.exe $@
```

- Make sure to give the shell scripts executable permissions, using:

```
chmod +x ./quasar
chmod +x ./redshift
```

- run "`./quasar -version`". The output should look like:

```
./quasar -version
-------------------------------------------------------------------------------
Quasar 1.0.0.0 - build date 11/20/2012 1:29:12 PM - 32-bit
Copyright © Bart Goossens 2011-2012
-------------------------------------------------------------------------------
OS: MacOS - Unix 10.8.0.0
CPU - unknown processor [2 cores]
CUDA (driver version: 4.2, runtime version: 4.2, 32-bit)
GPU #0 - GeForce 9400M [Mem: 253 Mb, Compute cap.: 1.1, 2 cores]
Clock rate: 1100000 Hz
Can map host memory: 1
Supports concurrent kernels: 0
Integrated with CPU memory: 1
```

- If this succeeds (you get the version info), you can just go ahead and run the Samples! For example:

```
./quasar -debug Samples/imshow.q
```

- If `imshow.q` works, this doesn't mean more advanced (compilable) samples will work.

5 Known issues

5.1 Windows Remote desktop connection

On Windows, Quasar cannot run using the GPU computation engine, when the GPU is the primary GPU of the system. Instead, either:

- Run Quasar using the CPU engine (`-cpu` command line option).
- Set up a VNC server on the windows machine, and connect through a VNC client.

A better solution is to install Microsoft TeamViewer Host on the server (with the GPUs), and to run Microsoft TeamViewer on the client. Although Microsoft TeamViewer uses standard HTTP/SSL ports, it is also possible to use SSH tunneling. The TeamViewer option works correctly in combination with OpenGL and is hence the recommended solution for using Quasar/Quasar Redshift via a remote connection.

5.2 Windows: problems starting Quasar / Redshift

1. When starting `Quasar.Redshift.exe` (or `Quasar.Spectroscope.exe`) leads to the error message:

   ```
   An unhandled exception of type 'System.IO.FileNotFoundException'
   occurred in Unknown Module. Additional information: Could not
   load file or assembly 'glib-sharp, Version=2.12.0.0, Culture=neutral,
   PublicKeyToken=35e10195dab3c99f' or one of its dependencies.
   The system cannot find the file specified.
   ```

   You can solve the problem by copying the DLLs from the folder NewestVersion\OS_Runtimes\32bit_Windows (note: not 64bit_Windows) to NewestVersion\. If this does not work for you, inspect the file "error.log" which has additional error information. You can send this file to me.

5.3 Windows: laptops with NVidia Optimus

Some display functions may not work by default on laptops with NVidia Optimus (i.e. laptops that both have an Intel HD Graphics GPU and an NVidia GPU). The solution is to create an application profile with the NVidia Control Panel for `Quasar.exe` and `Quasar.Redshift.exe` (and optionally `Quasar.Spectroscope.exe`). For "select the preferred graphics processor for this program", choose "High-performance NVIDIA processor".

5.4 Linux/MAC OS terminal server connection (ssh)

When using plots or forms, please ensure that X-window forwarding is enabled. This can be done as follows:

```
ssh -X [user@computername]
```

Quasar attempts to detect terminal server sessions. In this case, OpenGL interoperability with CUDA (which enables visualization directly from GPU memory, instead of copying data back to the CPU) cannot be used. By default, the OpenGL functionality will be disabled. However, some cases have been reported in which the terminal server session detection fails. To solve this problem, OpenGL functions can be disabled manually, by passing the `-nogl` command line option to Quasar).

5.5 Linux: Window managers

Some window managers with display effects (hence using the GPU), may cause CUDA to malfunction. Quasar programs generate failure messages such as "CufftSetupFailed" or hang during startup. The solution is then simply to use another window manager (for example, GNOME classic without effects).

5.6 MAC OS X: OpenGL / imshow does not work

It may be necessary to disable OpenGL. Edit `Quasar.Runtime.config.xml` and set the value for key "`USE_OPENGL`" to `False`.

5.7 Multiple NVidia GPUs: SLI mode on/off

It is recommended to turn *off* the SLI mode in the NVidia driver, so that Quasar can use the memories of both GPUs in an efficient manner.

6 Other issues

Contact us at quasar@telin.ugent.be.

Happy Quasar'ing!