Instructions for Building `QRFactor`

This document will outline the settings and libraries required to build and execute QRFactor in either Visual Studio (Windows) or a Docker container (Linux). It is assumed that Docker and/or Microsoft Visual Studio has been installed. If using Visual Studio, the CUDA Toolkit will also need to be downloaded separately. If using Docker, a docker account is required to pull from Docker Hub. The read-in of the system matrix A requires the data to be in Matrix Market format (see the NIST page for more information on MMF, as well as to download the library routines `mmio.h` and `mmio.c`).

1 Running `QRFactor` in Visual Studio 2019 (VS2019)

Place the system matrix file `sysMatA.mtx`, the library routines `mmio.c` and `mmio.h`, the wrapper `mmio_wrapper.cpp`, the source file `QRFactor.cpp`, and the VS2019 files `QRFactor.vcxproj` and `QRFactor.vcxproj.user` in a single directory. The time-dependent input data files can be located in a separate directory that will be set later.

There are several project settings that will need to be changed before the program can be built. These will be discussed below, and screen shots of the relevant settings will be included in appendix A. The following settings will ensure that VS2019 is able to find the correct library files:

- Under Advanced Properties, ensure **MSVC Toolset** is set to 14.25.28610.
- Under C++/General, ensure **Additional Include Directories** contains ./;$(CudaToolkitIncludeDir);$(CudaToolkitIncludeDir)/include;C:/ProgramData/NVIDIA Corporation/CUDA Samples/v10.2/common/inc.
- Under C++/Optimization, ensure **Optimization** is set to Maximum Optimization (Favour Speed) (/O2).
- Under Linker/General, ensure **Additional Library Directories** is set to $(CudaToolitLibDir).
- Under Linker/Input, ensure **Additional Dependencies** includes cusolver.lib;cusparse.lib; cudart_static.lib;kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;advapi32.lib; shell32.lib;ole32.lib;oleaut32.lib;uuid.lib;odbc32.lib;odbccp32.lib;%(AdditionalDependencies).

With these settings in place, the project can be built by selecting Build → Build QRFactor (Ctrl + B). `QRFactor.exe` can then be run either through VS2019 or via the command line.

2 Running `QRFactor` in a Docker Container

To create the appropriate container, see the instructions in § B. Once again, the system matrix file `sysMatA.mtx` – as well as auxillary files `mmio.c`, `mmio.h`, and `mmio_wrapper.cpp` – and source file `QRFactor.cpp` should all be placed in the same directory. Time-dependent input files can be placed either in the same directory, or in a different directory.

Compilation will take place using a `Makefile`. As a basis, the Makefile provided in the Toolkit sample directory `samples/7_CUDALibraries/cuSolverSp_LowlevelQR` can be copied into the `QRFactor` directory. The following changes to that Makefile need to be made:

- `ALL_CCFLAGS` should include an optimization level. To add this, insert a line after `ALL_CCFLAGS :=` that reads `ALL_CCFLAGS+= -O2`, or the desired level of optimization.
- The location of helper libraries from the CUDA toolkit need to be specified by changing the INCLUDES variable from `INCLUDES := -I../../common/inc` to

  ```
  INCLUDES := -I$(CUDA_PATH)/samples/common/inc
  ```

  as well as adding an additional line that reads

  ```
  INCLUDES+= -I$(CUDA_PATH)/targets/x86_64-linux/include
  ```

  or the equivalent architecture.

With these changes to the `Makefile`, the program is compiled by running `make` as per usual.

A Visual Studio Settings

Included below are screen shots of the options discussed in § 1 for running `QRFactor` in VS2019.

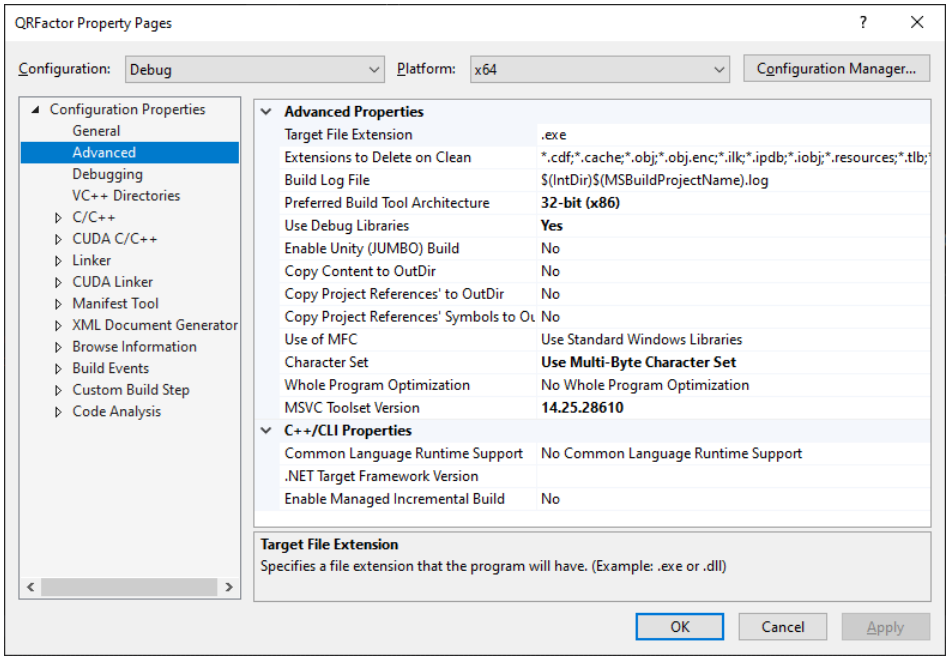

Figure 1: QRFactor Properties → Advanced

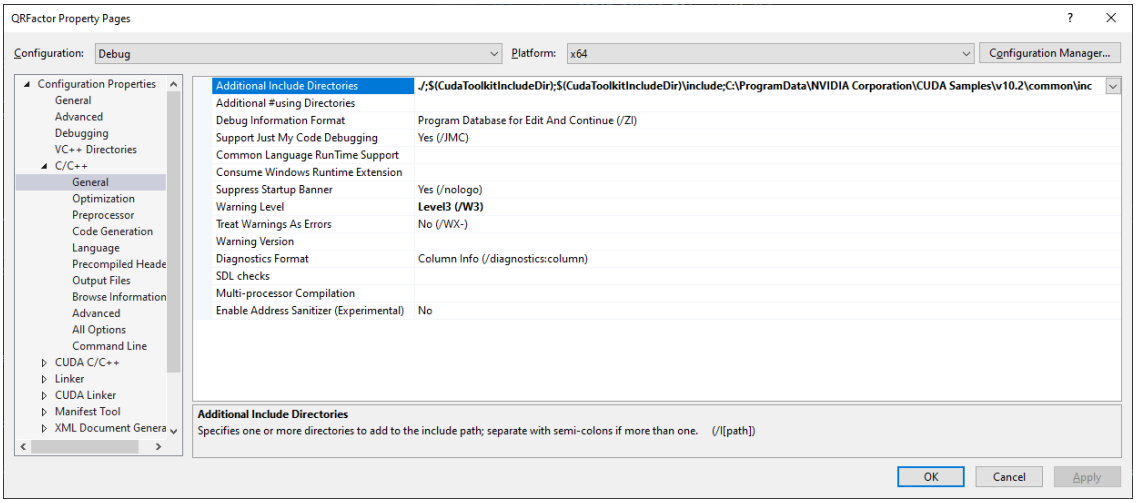

Figure 2: QRFactor Properties → C/C++ → General

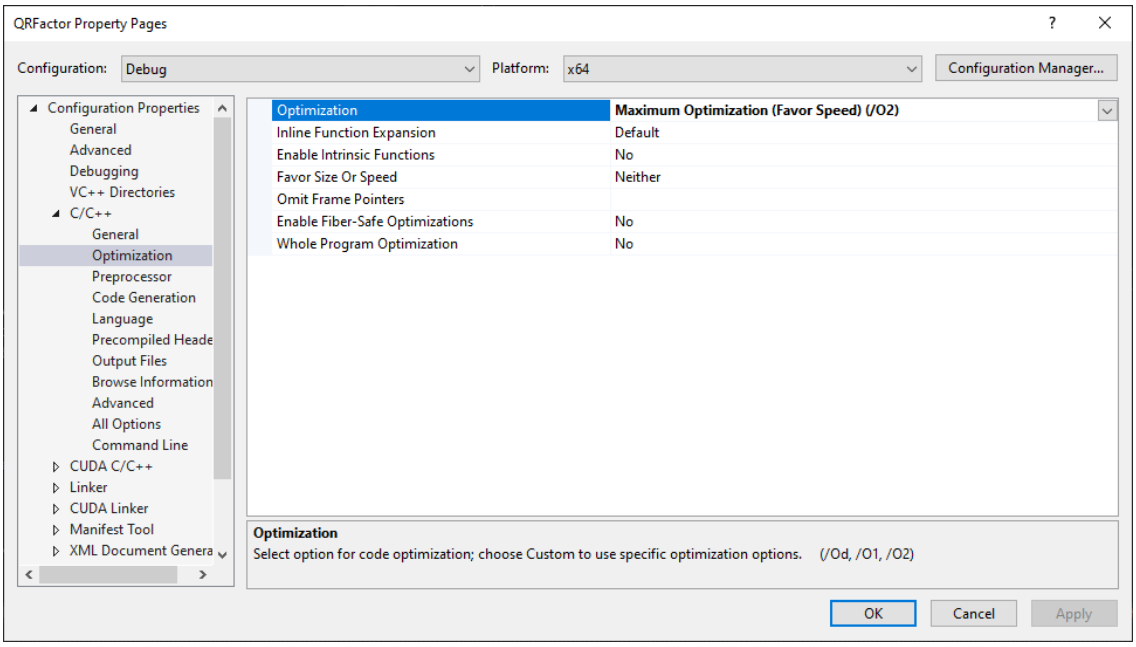

Figure 3: QRFactor Properties → C/C++ → Optimization

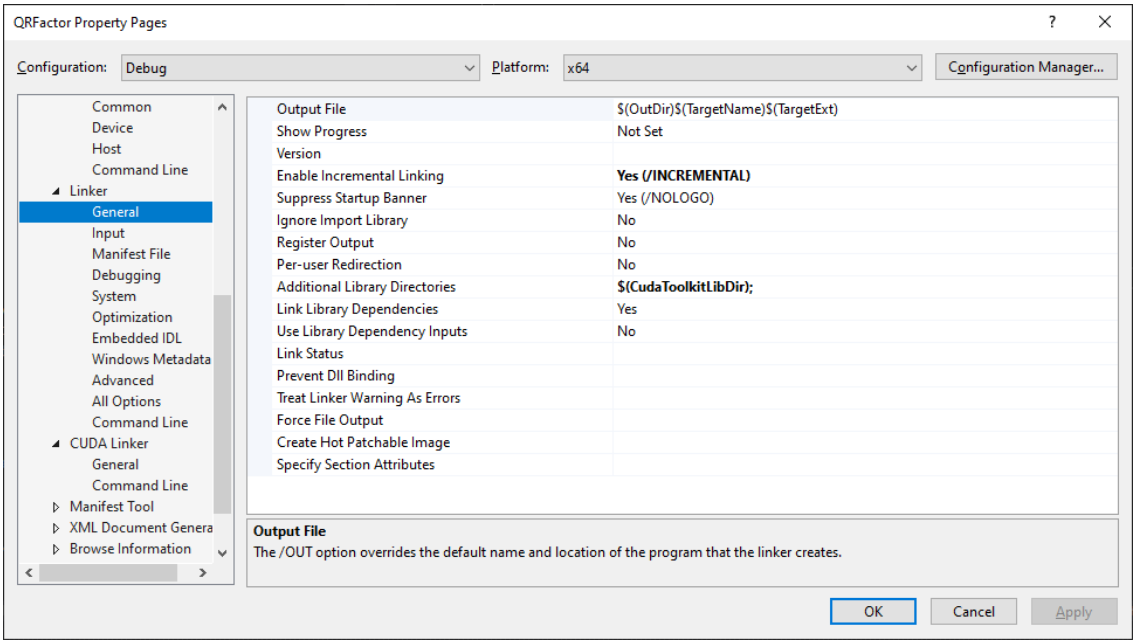

Figure 4: QRFactor Properties → Linker → General

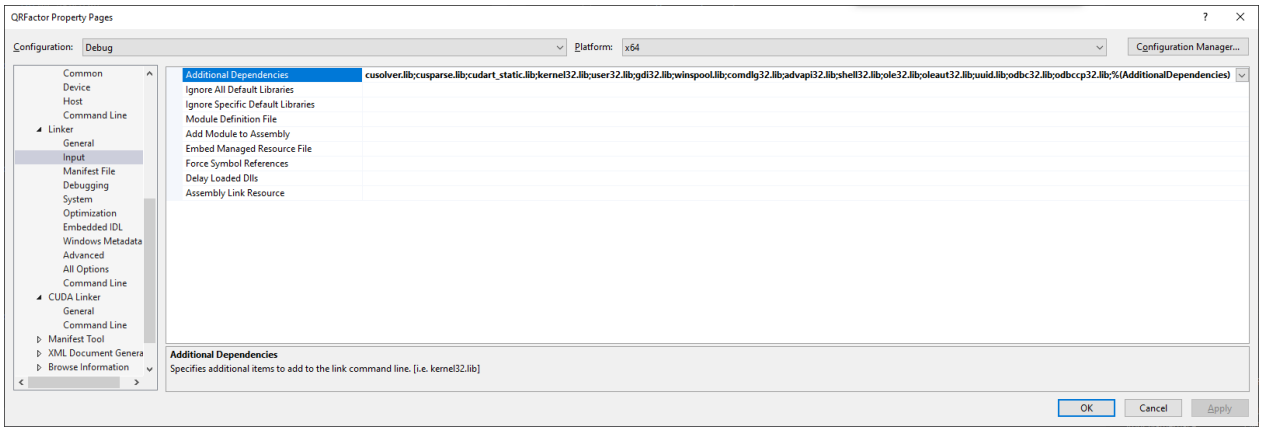

Figure 5: QRFactor Properties → Linker → Input

B Docker

Included below are the steps required to create a custom CUDA image that includes the CUDA Toolkit within a linux-based Docker container. Also included are sample commands for running a container based on that image.

1. In the desired directory, create a file named 'Dockerfile' with no extension. This file will include instructions for creating an image that will provide the basis for the development environment. The Dockerfile is provided in § C. To make the image, run

   `docker build -t <CUDA_IMAGE> .`

   where `<CUDA_IMAGE>` is the name chosen for the image that is being created.

2. Allow Docker to run through the steps involved in creating the image – this should not require any input from the user. This process can also be lengthly, but will only need to happen once.

3. Confirm the image has been created by running `docker images ls` and confirming that `<CUDA_IMAGE>` was created.

4. Edit the contents of `runCUDA.sh` (provided in § D) to set `<CUDA_IMAGE>` to the image that was just created, and `<CUDA_CONTAINER>` to the name chosen for the container that development will occur within. Volumes can be mapped into/out of the container using the `-v` option, as shown. Running `bash runCUDA.sh` will create and enter `<CUDA_CONTAINER>`.

5. The container can be exited without stopping by pressing `Ctrl+q, Ctrl+p` and reentered later by running `docker attach <CUDA_CONTAINER>`.

6. To perminantly delete the container or image, run `docker rm <CUDA_CONTAINER>` or `docker rmi <CUDA_IMAGE>`.

C Dockerfile

```
FROM nvidia/cuda:10.2-base-ubuntu18.04

# Package management functions
RUN apt-get update
RUN apt-get install -y apt-utils && \
apt-get install -y wget && \
apt-get install -y gnupg && \
apt-get install -y curl && \
apt-get install -y linux-headers-$(uname -r)

# Get CUDA Toolkit (see https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html)
RUN wget https://developer.download.nvidia.com/compute/cuda/
                                        repos/ubuntu1804/x86_64/cuda-ubuntu1804.pin
```

```
RUN mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600
RUN wget http://developer.download.nvidia.com/compute/cuda/10.2/Prod/local_installers/
                  cuda-repo-ubuntu1804-10-2-local-10.2.89-440.33.01_1.0-1_amd64.deb
RUN dpkg -i cuda-repo-ubuntu1804-10-2-local-10.2.89-440.33.01_1.0-1_amd64.deb
RUN apt-key add /var/cuda-repo-10-2-local-10.2.89-440.33.01/7fa2af80.pub
RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y cuda

# Install samples that can be executed
RUN cuda-install-samples-10.2.sh /home/cuda-samples

# Clean
RUN apt-get clean && rm -rf /var/lib/apt/lists/* /temp/* /var/temp/*

# Update PATHs
RUN export PATH=/usr/local/cuda-10.2/bin:/usr/local/cuda-10.2/
                                                    NsightCompute-2019.1${PATH:+:${PATH}}
RUN export LD_LIBRARY_PATH=/usr/local/cuda-10.2/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

D runCUDA.sh

```
# Run a docker container with cuda installed and gpus active
docker run -it --gpus all \
#    -v /home/bradc/MHI/Upload:/home/data \
#    -v /home/bradc/MHI/QRFactor:/home/QRFactor \
     --name <CUDA_CONTAINER> \
    <CUDA_IMAGE>
```