The University of Birmingham
School of Computer Science


# *NSight — Getting Started*

These instructions are for setting up and getting started with the NSight IDE for C programming on CPUs (NOT GPUS) in the school labs on Linux systems.

NSight is part of the NVidia CUDA Development Kit, and though we are not going to use CUDA for C programming on CPUs, we still need to set the necessary environment variables to get proper access to NSight. These instructions are ONLY for the School computers. Run:

```
module load cuda
```

`nsight` is a version of the `Eclipse` IDE modified by NVidia for CUDA project development, although it also works for ordinary C or C++ development. Like `Eclipse`, `nsight` uses a workspace directory (by default `~/cuda-workspace`) and, when you create a project, offers to store your project files there. I recommend that you do **NOT** do so, but instead that you put your project files somewhere outside the workspace directory. A good approach is to make a directory called "`cuda`" somewhere in your school GIT repository (you can make a new repository if you wish to keep it separate from your other repositories). This way you can easily delete the `~/cuda-workspace` directory if you mess up your `nsight` settings without losing your projects. Also it means you can easily work at home on your own machine sharing your project files through GIT.

First we will import one of the sample applications and compile and run it:

1. Download the sample programs from the Canvas page of this module and unzip them into some reasonable part of your filesystem. The following instructions shows how to setup "`C2-imflipP.zip`" as an nsight project
2. Start `nsight`
3. Select `File|New|C Project`
4. Set the project name to "`C2-imflipP`", untick "`use default location`", set the location to wherever you unzipped the "`C2-imflipP.zip`" file.
5. In the "`Project type:`" window, select "`Empty Project`
6. In the "`Toolchains`" window, select "`Linux GCC`"
7. Click on "`Next`"
8. Make sure both "`Debug`" and "`Release`" configurations are selected and click "`Finish`"
9. Now add the "`pthread`" library: Select `Project|Properties`.
10. On the left side of the dialog box, select `C/C++ General|Paths and Symbols`
11. Select the "`Library`" tab and click on "`Add`" on the right hand side
12. Enter "`pthread`", (without the quotes), into the "`File`" box in the dialog box, tick the tickbox "`Add to all configurations`" and click on "`OK`" and then on "`OK`" in the previous dialog box
13. Click on the hammer icon on the button bar to build the debug version (you can select, using the drop-down arrow to the right of the icon whether you want to build the debug or the release version - we want the debug version for now, but you might as well build the release version now as well. If you do, make sure you switch back to the "`Debug`" version, or you will stay working on the release version.
14. Now click on `Run|Run Configurations...`
15. Double click on "`C/C++ Application`"
16. Select the new "`C2-imflipP Debug`" line. Append on the end of the "`Name`" the characters "` V 1`"
17. Select the "`Arguments`" tab and enter "`images/townSquare.bmp images/townSquareV.bmp V 1`"
18. Click "`Apply`", then "`Run`" and the program should run with output to the "`Console`" tab of the bottom window.
19. You can now re-run by clicking on the green run button in the button bar.
20. You can add more `Run Configurations` similarly, for `H` as well as `V` flips, and for more threads. In each case, make sure that the name of the configuration matches the argument string.

## 1 Exercises

a. First just get the 3 projects working

b. Make a table of the timings for `H` and `V` flips against different numbers of threads: from 1 up to twice the number of threads that the CPU supports. Annotate the table with the CPU Model, the nuber of cores and threads that it supports, and the sizes of the L1, L2 and L3 caches. Do this for as many machines with different CPUs that you have access to.

c. In the "`C3_imflipPM`" project, extend your table to include the `W` and `I` flips as well.

d. Start a new project based on "`C2_imflipP`' and modify to be a 90 degree rotate. This means that instead of swapping two pixels, you will need to move pixel values on one step through a rotating group of 4 pixels. Get this working for the serial case first, then make it work for more than one thread.

e. If you really want to practice your C programming skills: Fix the various imflip programs so that they work on non-square images.