|  |
| --- |
| Centre_for_Water_Systems_rgb.jpg |
| CADDIES:  Source Code |
| BUILD INSTRUCTIONS |
|  |
| **Michael J Gibson, Michele Guidolin, Albert S. Chen, Nicola Pasquale** |
| **April 2018** |

|  |
| --- |
| Application Version 120 |

****

# Introduction

The **caflood** application performs a 2D pluvial flood inundation simulation using cellular automata (CA) techniques instead of solving the classic shallow water equations (SWEs). The CA technique offers a versatile method for modelling complex physical systems using simple operations (Wolfram 1984). This simplification dramatically reduces the computational load of a CA model in comparison to a physically based model.

CA model usually consists of five essential features: a discrete space, the distribution of the neighbour cells, the state of the cells, the discrete time step and the transition rules (Itami 1994). The transition rules are composed of simple operations that govern the evolution of each cell’s state. This makes use of the previous state of the cell itself and those in its neighbourhood. Since computing the new state of a cell does not depend on the state of any other cells at the same time step, CA algorithms are well suited to parallel computation.

The caflood application is part of the **CADDIES** framework/project which is final aim is to produce faster algorithms for handling dual drainage flood modelling, i.e. where the urban surface flow (major system) is combined with the sewer flow (minor system). This is achieved by using CA techniques together with modern parallel hardware.

|  |
| --- |
| ca-2d-1d.png  Figure 1. Example of interactions between 2D surface flow and 1D sewer flow via linking elements (manholes) |

The caflood application implements the Weighted Cellular Automata 2D model (**WCA2D**) with simplified transition rules to simulate inundation events instead of using complex physically based equations and mathematical operations. This model improves the methodology adopted in the CA2D model (Ghimire et al. 2013) using a weight-based system. The WCA2D is a diffusive-like model that ignores inertia terms and momentum conservation. The model has been designed to work with various general grids, (e.g., rectangular, hexagonal or triangular grid) with different neighbourhood types (e.g., the four cells of the von-Neumann (VN) neighbourhood or the eight cells of the Moore neighbourhood).

The major points of this model are:

* The ratio of water transferring from the central cell to the downstream neighbour cells (intercellular-volume) is calculated using a minimalistic and quick weight-based system;
* The volume of water transferring between the central cell and the neighbour cells is limited by a single equation, which is a composition of a simplified Manning’s formula and the critical flow condition.
* Both the adaptive time step and the velocity, as an average velocity, are evaluated within a larger update time step to speed up the simulations.

Additional information about the WCA2D model is available in (Guidolin et al. 2015)

The caflood application uses the **CADDIES Application Programming Interface** (API) to implement the 2D model (Guidolin et al. 2012). The CADDIES API defines a standard set of methods, data structures and variables that can be used to develop parallel CA algorithms. A developer needs to write the code of the CA model only once. After that, the CADDIES API gives the flexibility to produce the same CA model for any type of CA grid, square/hexagonal/triangular grid, and to use different high performance acceleration techniques without changing the code or with minimum effort.

In caflood, the WCA2D model has been implemented using only a square cell grid with von-Neumann neighbourhood. Furthermore, thanks to the CADDIES API, a simulation can be executed in a multi-core CPU using OpenMP library (Dagum and Menon 1998) and in a multi-core CPU and on a graphics card GPU using the OpenCL library (Munshi and others 2011).

# locating the source code

There two main locations/ways to acquire the source code for the CADDIES frameworks, either at the University of Exeter CWS site, where zip files for the API, each implementation (simple serial, OpenMP and OpenCL), and caFlood applications can be located separately:

<http://emps.exeter.ac.uk/engineering/research/cws/resources/caddies-framework/caddies-download/>

Alternatively the source code has now been migrated to an online GIT repository, located at:

<https://git.exeter.ac.uk/caddies>

If you download the API from the university website, it will come in several zip files:

* The base directory structure, cmake and common files : CADDIES-base-120.zip
* The caFlood application : CADDIES-app-caddies-caflood-120

A number of implementations, currently using a square grid, single level radius of Von Neumann neighbourhood, of which at least one is required:

* Simple serial : CADDIES-CAAPI-impl-S1LVNSG-120.zip
* Parallel CPU/shared memory – OpenMP : CADDIES-CAAPI-impl-OMP1LVNSG-120.zip
* Parallel GPU/many-core – OpenCL : CADDIES-CAAPI-impl-OCL1LVNSG-120.zip

In order to build the caFlood application, the base, and at least one (or possibly all) implementations should be downloaded and extracted to a common directory. Finally the caFlood application can also be extracted the same directory.

Downloading the GIT respoitory the API and caFlood applications can be downloaded; note to place the caFlood application in the sub directory CADDIES/apps.

The CADDIES framework is built using Cmake, which is available from:

<https://cmake.org/>

# build instructions – windows

Having downloaded the necessary files, shown in section 2; in this example, we will use the directory:

C:\prog\caddies

Create another directory, which will contain the built versions of the API/Application, in this example, the following directory is utilised:

C:\prog\caddies\_builds

Run CMake GUI, and inform the program of the location of the source code, and build directory. It is actually advisable to have multiple sub directories for the different possible builds.

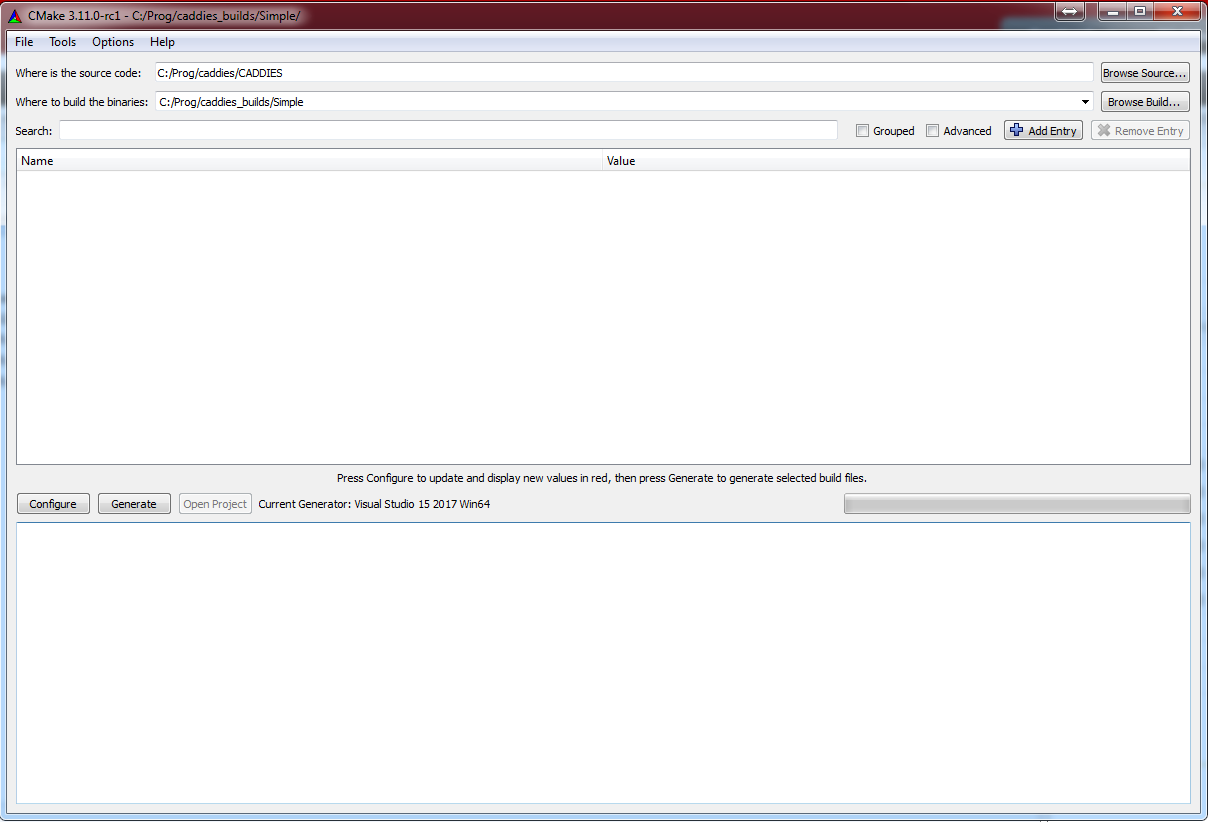


Figure 2 – Example of cmake GUI on windows

The next step is to click the *configure* button, which will pop-up another window which will allow the user to select the compiler/SDK, and including if this is a 32bit of 64bit build, shown in Figure 3. In the example in Figure 3, the Microsoft visual studio 15 2017 is selected with 64bit build. Finally, click finish, after which Cmkae will search and locate your compiler/SDK, and configure the build, and the result should look something like Figure 4.

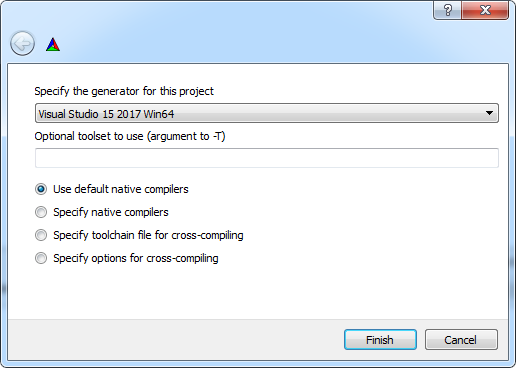


Figure 3 – Cmake pop-up compiler/SDK selection window example



Figure 4 – Example Cmake configuration

Configuration settings of the build maybe now be applied in the highlighted area:

* CAAPI\_REAL\_PRECISION : (either float or double) sets the floating point precision used
* CAAPI\_SPECIFIC\_IMPL\_DIR : one of the implementation sub directories, defaults to simple implementation

If the OpenCL implementation is selected, the Cmake script will search for the OpenCL libraries and include directories on your system, although it is quite possible that it may not be able to find these, and may therefore pop-up an error message box shown in Figure 5.

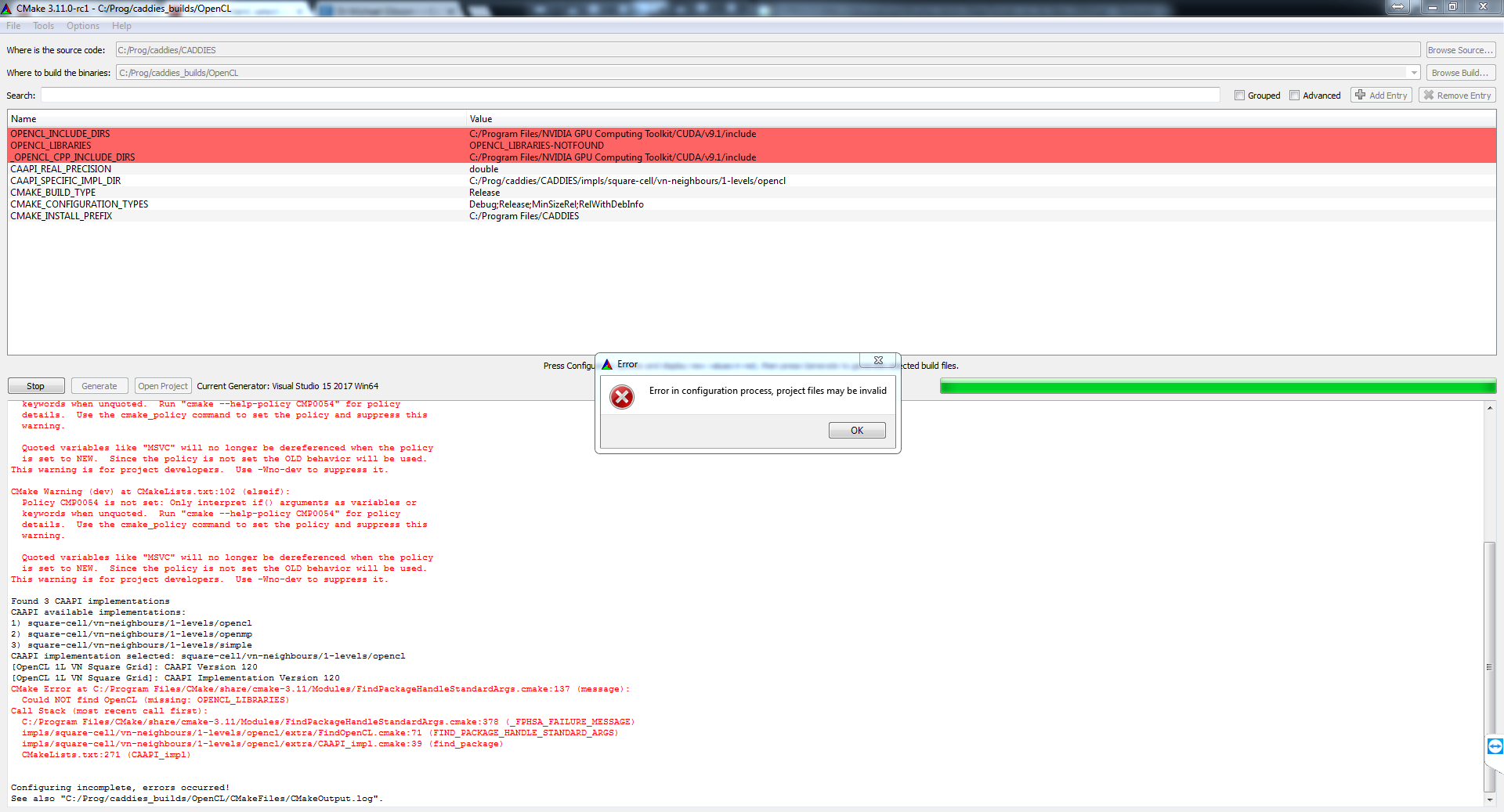


Figure 5 – Example of OpenCL implementation initial configuration

It is advised to set the \_OPENCL\_CPP\_INCLUDE\_DIRS to the *extra* folder within the OpenCL implementation folder to avoid conflicts with differing versions of the include files. Finally located the OpenCL dynamic link library (DLL) file, to which the project will be linked. Configuration settings specific to the OpenCL implementation may now be set for this build:

* CAAPI\_OCL\_EVENTS : (enable/disable) actives or not, the OpenCL events.
* CAAPI\_OCL\_NATIVE : (enable/disable) activates or not, the OpenCL native functions
* CAAPI\_OCL\_TABLE : (constant/global) specific the memory type to use for tables

It is noted that Nvidia implementations do not work well with OpenCL events, and can cause large memory leaks, and should therefore be disabled (*disable*). However Intel implementations do not work well without OpenCL events, and should therefore be enabled (*enable*).

Finally, clicking generate will make Cmake create a SDK solution ( for the likes of Microsoft visual studio), or compile directly for other compilers. Load the ALL\_Build solution and compile.

For further assistance, please contact Dr Michael J Gibson : [M.J.Gibson@Exeter.ac.uk](mailto:M.J.Gibson@Exeter.ac.uk)

# References

Dagum, Leonardo, and Ramesh Menon. 1998. ‘OpenMP: An Industry Standard API for Shared-Memory Programming’. *Computational Science & Engineering, IEEE* 5 (1): 46–55.

Ghimire, Bidur, Albert S. Chen, Michele Guidolin, Edward C. Keedwell, Slobodan Djordjević, and Dragan A. Savić. 2013. ‘Formulation of a Fast 2D Urban Pluvial Flood Model Using a Cellular Automata Approach’. *Journal of Hydroinformatics* 15 (3): 676. doi:10.2166/hydro.2012.245.

Guidolin, Michele, Albert S. Chen, Bidur Ghimire, Ed Keedwell, Slobodan Djordjevic, and Dragan A. Savic. 2015‘A Weighted Cellular Automata 2D Inundation Model for Rapid Flood Analysis’. *Under Review*.

Guidolin, Michele, Andrew Duncan, Bidur Ghimire, Mike Gibson, Edward Keedwell, Albert S. Chen, Slobodan Djordjevic, and Dragan Savic. 2012. ‘CADDIES: A New Framework for Rapid Development of Parallel Cellular Automata Algorithms for Flood Simulation’. In Hamburg, Germany: IWA (International Water Association).

Itami, Robert M. 1994. ‘Simulating Spatial Dynamics: Cellular Automata Theory’. *Landscape and Urban Planning* 30 (1): 27–47.

Munshi, A., and others. 2011. ‘The OpenCL Specification Version 1.1’. *Khronos OpenCL Working Group*.

Wolfram, Stephen. 1984. ‘Cellular Automata as Models of Complexity’. *Nature* 311 (October): 419–24.