Neal, J., Dunne, T., Sampson, C., Smith, A., & Bates, P. (2018). Optimisation of the two-dimensional hydraulic model LISFOOD-FP for CPU architecture. *Environmental Modelling and Software*, *107*, 148-157. https://doi.org/10.1016/j.envsoft.2018.05.011

Environmental Modelling and Software 107 (2018) 148–157

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

Optimisation of the two-dimensional hydraulic model LISFOOD-FP for CPU architecture

Jeffrey Neal[a,*], Toby Dunne[a], Chris Sampson[b], Andrew Smith[b], Paul Bates[a]

[a] School of Geographical Sciences, University of Bristol, Bristol, BS8 1SS, UK
[b] Fathom, Engine Shed, Temple Meads, Bristol, BS1 6QH, UK

ARTICLE INFO

Keywords:
Flood inundation modelling
HPC
LISFLOOD-FP
Parallelisation
Vectorisation

ABSTRACT

Flood inundation models are increasingly used for a wide variety of river and coastal management applications. Nevertheless, the computational effort to run these models remains a substantial constraint on their application. In this study four developments to the LISFLOOD-FP 2D flood inundation model have been documented that: 1) refine the parallelisation of the model; 2) reduce the computational burden of dry cells; 3) reduce the data movements between CPU and RAM; and 4) vectorise the core numerical solver. The value of each of these developments in terms of compute time and parallel efficiency was tested on 12 test cases. For realistic test cases, improvements in single core performance of between 4.2x and 8.4x were achieved, which when combined with parallelisation on 16 cores resulted in computation times 34-60x shorter than previous LISFLOOD-FP models on one core. Results were compared to a sample of commercial models for context.

1. Introduction

Predictions of flood hazard from two-dimensional flood inundation models form an essential component of flood risk management strategies in many countries (de Moel et al., 2009). The use of these models has increased substantially over the last 20 years due, in part, to an increase in the availability of precise and accurate Digital Terrain Models (DTM). Datasets with sub meter resolution are becoming increasingly available in urban areas where fine resolution is needed to capture the complex flow pathways around urban structures (Schubert and Sanders, 2012) or resolve small scale flow connectivity (Neal et al., 2011; Yu and Lane, 2006). This need for high resolution inundation simulation results in a situation where computational resource becomes one of the main factors affecting simulation accuracy in practical applications. Two dimensional inundation models are also increasingly used at large scale for modelling of globally significant wetland systems (de Paiva et al., 2013; Yamazaki et al., 2011), providing continental overviews of flood hazard (Alfieri et al., 2014; Dottori et al., 2016; Sampson et al., 2015; Vousdoukas et al., 2016) or as the surface water flow component of landscape evolution modelling systems (Adams et al., 2017b; Barkwith et al., 2015; Coulthard et al., 2013). For these applications, the size of the domain and the requirement to characterise model uncertainty through Monte Carlo simulation also creates significant computational cost, where thousands of simulations can be necessary to explore the models parameter space.

A substantial body of work has been undertaken to address these issues, with solutions falling into two categories: 1) Developments to the governing equations that improve the numerical schemes; and 2) Parallelisation of the code for application on multiple computational cores. Developments to the governing equations are wide ranging but often include simplification of the physical process representation such as the removal of inertia terms from the shallow water equations (Bates et al., 2010; de Almeida et al., 2012; Dottori et al., 2016), or the omission of any floodplain dynamics (Gouldby et al., 2008; Winsemius et al., 2013). The limitation of such an approach is that as the models become simpler the range of applications where they are applicable and simulation accuracy typically reduces (Vousdoukas et al., 2016). By contrast, parallelisation does not change the model simulation and typically involves implementation of the model over multiple processors via message passing (Neal et al., 2010; Sanders et al., 2010), threading on shared memory central processing units (Judi et al., 2011; Leandro et al., 2014; Neal et al., 2009a; Petaccia et al., 2016) or by offloading work onto Graphical Processing Units (GPUs) (Kalyanapu et al., 2011; Lamb et al., 2009; Petaccia et al., 2016; Vacondio et al., 2017). However, as technology continually develops it becomes periodically necessary to revisit the optimisation of these numerical schemes in order to benefit from the enhanced capabilities of new hardware. It is also necessary to understand the potential benefits of undertaking code development work and if perceived improvements to the code are realised across a wide range of realistic test scenarios.

* Corresponding author.
E-mail address: j.neal@bristol.ac.uk (J. Neal).

https://doi.org/10.1016/j.envsoft.2018.05.011
Received 28 July 2017; Accepted 21 May 2018
Available online 22 May 2018

This paper revisits the parallelisation of a numerically efficient two-dimensional flood inundation model (LISFLOOD-FP) on multicore ×86 CPU processors to investigate what changes to the code structure are most beneficial in order to utilise recent developments in CPU architecture. In addition to substantial refinements to the parallelisation of the model, we document the impact of vectorising the numerical scheme, adapting how the code processes the model domain such that only wet cells are evaluated and writing the code to allow for better memory management by the compiler. The performance of the model was evaluated using a range of test cases that are representative of typical inundation modelling applications. Since a substantial number of flood inundation modelling codes exist we hope that this short paper will provide useful information for researchers and practitioners developing their own model.

2. Model description

The LISFLOOD-FP code was used as the hydraulic model in this study, but is typical of a wide range of similar schemes. The model solves the shallow water equations, without the convective acceleration term, on a staggered Cartesian grid using an explicit finite difference method. Numerically, this involves calculating the flow between cells given the mass in each cell (momentum equation, eq. (1)) and the change in mass in each cell given the flows between cells (continuity equation, eq. (2)). These equations, including their derivation, are reported in detail elsewhere (Bates et al., 2010; de Almeida et al., 2012) and are therefore only briefly outlined here. The momentum equation is described by:

$$Q_{i+1/2}^{t+\Delta t} = \frac{Q_{i+1/2}^{t} - gA_{flow}^{t}\Delta t S_{i+1/2}^{t}}{1 + g\Delta t n^2 |Q_{i+1/2}^{t}| / [(R_{flow}^{t})^{4/3} A_{flow}^{t}]} \quad (1)$$

Where $Q_{i+1/2}^{t+\Delta t}$ is the flow rate in between two cells i and $i+1$ that will apply from time t to $t+\Delta t$, A_{flow}^{t} is the area of flow between cells, R_{flow}^{t} is the hydraulic radius, $S_{i+1/2}^{t}$ is the water surface slope between cells, n is Manning's roughness coefficient and g is acceleration due to gravity. For each cell the momentum equation is implemented at all four interfaces with its neighbours before applying the continuity equation to the cell:

$$V_{i,j}^{t+\Delta t} = V_{i,j}^{t} + \Delta t (Q_{i-1/2,j}^{t+\Delta t} - Q_{i+1/2,j}^{t+\Delta t} + Q_{i,j-1/2}^{t+\Delta t} - Q_{i,j+1/2}^{t+\Delta t}) \quad (2)$$

Where V is the cell volume from which water surface elevation is easily computed, while i and j index the Cartesian grid. The model also includes subroutines to simulate rainfall, routing of flows over steep surfaces (Sampson et al., 2013), 1D river channels (Neal et al., 2012a), evaporation from open water and some hydraulic structures (Bates et al., 2016). Details of these are available in the LISFLOOD-FP user manual (Bates et al., 2016). In practical terms, the calculation stencil for the momentum equation never exceeds the two neighbouring cells, while the continuity equation stencil requires only the four adjacent flow estimates plus any source terms (e.g. rainfall, evaporation, runoff). Therefore, the domain can be easily decomposed and run on separate cores, making the scheme simple to parallelise as demonstrated by previous studies (Neal et al., 2010).

The left hand side of Fig. 1 describes the sequence of operations used by LISFLOOD-FP after it was parallelised and presented by Neal et al. (2009a). For the purpose of this paper this will be called "original" LISFLOOD-FP and represents the basic architecture of all LISFLOOD-FP versions between Neal et al. (2009a) and this paper. This code architecture is a logical way of solving the governing equations and we would imagine can be widely adopted. After reading the necessary input data and parameters from disk this version of the model simulates the hydrodynamics using five functions that each loop across the model domain (Fig. 1a) undertaking the following numerical operations: 1) calculate eq. (1) in the x direction between all cells; 2) calculate eq. (1) in the y direction for all cells; 3) implement a variant of eq. (1) along all model boundary cell edges; 4) add any source terms to the cells; and 5) implement eq. (2) for all cells. Each loop is easily parallelised as shown in the pseudo C code in Fig. 2, which is applicable to most explicit hydrodynamic models.

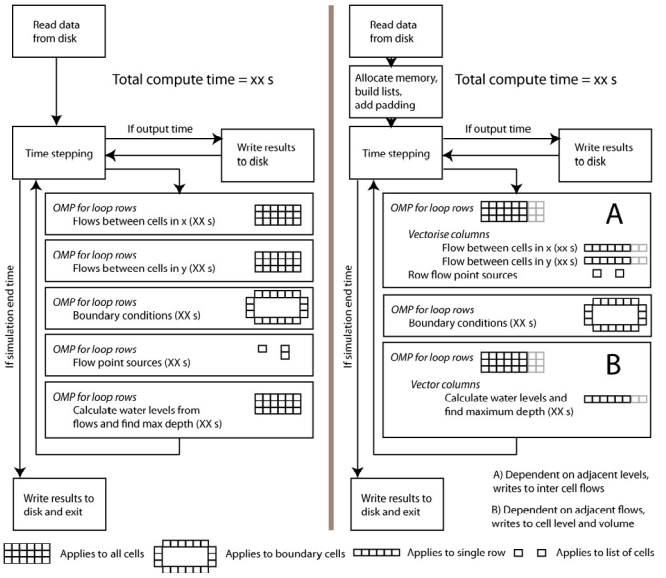

Fig. 1. Schematic describing the structure of the original (left) and optimised (right) versions of the code. The schematic describes a two-dimensional model domain with point source inflows and boundary conditions at edges.

Unfortunately, the layout of this code has a number of potentially significant limitations, the significant of which we will investigate in the results section, that may compromise computational efficiency and which can be summarised as:

1. Parallel loop structure
   - Each loop requires the creation of new threads that increase the overhead associated with parallelisation.
2. Wet and dry cells
   - A loop will access data for each cell regardless of whether that cell is wet or not e.g. on a dry domain data will be repeatedly accessed but no computation undertaken.
3. Data access
   - The loops repeatedly access the same DEM and parameter data from memory, meaning data must repeatedly be moved from RAM to the processor.
4. Vectorisation
   - The work within the loop is undertaken on a cell by cell basis and thus does not take advantage of potential vectorisation available on the processor.

2.1. Optimisation

The four issues above were addressed by making the following changes to the code, with the new structure summarised by the flow diagram in Fig. 1b.

2.1.1. Parallel loop structure

In the optimised code threads were created at the start of the simulation, rather than for each parallel *for* loop. The change is illustrated by the pseudo code in Fig. 3. Setting up the threads in this way is reasonably straightforward, however, unlike the situation where each loop is parallelised, all sections of the code that do not run in parallel need to be identified. Threads process rows of data in the model domain, with a *nowait* instruction used to let the compiler know that a thread can begin processing another row without waiting on other threads to finish. We assessed the parallel performance of the model by

```
#pragma omp parallel for
for(int i=0; i < number_of_rows_in_DEM; i++)
{
        for(int j=0; j < number_of_columns_in_DEM; j++)
        {
                // make some calculations
        }
}
```

Fig. 2. Example pseudo-code for parallel *for* loop in C.

comparing the original and optimised version of the model across a range of test cases using 1 to 16 threads.

2.1.2. Wet and dry cells

A simple tracking of the wet edge during an inundation simulation was implemented, which allows the numerical scheme to be active over a smaller portion of the model domain. This is by no means a new idea and the idea of tracking only wet cells has been around for some time. For example, the original JFLOW scheme (Bradbrook, 2006) maintains a look up list of wet and newly wet cells despite using a raster grid, while the CEASAR model adapts its active calculations in time i.e. by missing out periods of minimal dynamics (Coulthard et al., 2013). For each row of the model domain the cells are indexed from left to right in ascending order. When the simulation starts the wet cells with the lowest and highest index (*i_start* & *i_end*) are identified in each row, with *i_start* set greater than *i_end* when the row is dry. These indices are then expanded to align in memory when necessary (e.g. *i_start* might be reduced to fall on a memory block boundary) and used to define which cells in the row are considered by the numerical scheme. When a cell wets or dries a check is made to see if the indices need to be changed, and a check is also made for any source terms in the domain. The test cases in the results section will be used to assess the overhead of this scheme and its expected benefits for realistic simulation cases. One limitation of this simple approach occurs when dry cells are located between wet cells on a row. There is potentially an additional speedup to be gained over our approach by not visiting these cells, which would be done by expanding the algorithm to track multiple wet and dry edges per row. We have not assessed when such additional complexity could yield a faster simulation.

2.1.3. Data access

For most hydrodynamic models, the numerical effort required to calculate flow and update cell volume is such that the program will become inefficient if the computer has to do a lot of work moving data around in memory (Gibson, 2015; Leandro et al., 2014). By far the most significant change to the code was to rearrange the data access such that fewer movements of data between RAM and core cache are required. This is not something that the developer specifically controls but requires the code to be written in a structure that the compiler can more easily optimise. The most significant change to the structure made here was to combine the calculations of flow in x and y rather than have this arranged in two independent functions (see Fig. 1 box A) such that each cell is visited only once during the momentum calculation. The same applies for the continuity equation with respect to source terms such as evaporation and precipitation.

Furthermore, the original version of the code stores data for each variable (e.g. elevation, depth) as continuous blocks for the whole model domain. In the optimised version, the end of each row is padded such that the start of each rows data is 64 bit aligned, which allows the threads easier and quicker access to these data than is the case where rows can start anywhere in memory. These blocks are also numa aligned meaning the data are stored on RAM closest to the CPU where the computation will occur, reducing the need to move data between CPU sockets on the server.

2.1.4. Vectorisation

The final code development step was to vectorise the momentum and continuity equations for each row using advanced vector extension (AVX). As with improving how the code accesses data from RAM this is not explicitly controlled by the developer. Instead we rewrote the core computational component of the solver such that the compiler was able to implement the vectorisation. The code snippet in Appendix A describes the implementation of the momentum equation between two cells in a manner that can be vectorised on an Intel chip by the Intel

```
#pragma omp parallel default(shared)

#pragma omp for schedule(static) nowait

for (int block_index = 0; block_index < wet_dry_bounds->block_count; block_index++)
      {

#pragma omp single nowait

#pragma omp barrier // ensure all threads have finished their calculation of momentum
before proceeding to the continuity equation
```

Fig. 3. OpenMP structure.

Table 1
Model test cases: 2D = two-dimensional floodplain solver; SG = one-dimensional sub-grid scale river channel solver; E = evaporation; RR = Rainfall and Rainfall routing model; ST = structures (weirs). Res is the model resolution and t-steps is the number of simulation time steps needed to complete the simulation. Min-t is the minimum time step either fixed by the user (f) or variable based on the model stability criteria (v). Finally, N_bound is the number of boundary condition cells (at grid edges or as internal point sources). *The global flood model is a 16,300 km^2 area to the north west of Oklahoma City in the USA. **The pluvial test is a 17,500 km^2 area of central Scotland north of Edinburgh.

Test Case	Type	Rows	Cols	Cells (k)	Resolution	t- steps	Min-t	N_bound	Modules	Reference
Carlisle	Fluvial	611	951	581	5m	300831	0.38s (v)	100	2D	(Neal et al., 2009, 2013)
EA test 5 (Valley filling)	Fluvial	225	255	57	50m	49377	1.97s (v)	4	2D	(Neal et al., 2012b; Néelz and Pender, 2010)
Glasgow	Fluvial	200	480	96	2m	14143	0.46s (v)	1	2D	(Aronica et al., 2012; Fewtrell et al., 2008; Hunter et al., 2008)
Inner Niger Delta	Fluvial	426	603	257	905m	4.7M	45s (v)	3	2D + SG + E	(Neal et al., 2012a)
New York	Coastal	3452	2387	8240	10m	167416	0.35s (v)	1762	2D	(Duckworth, 2014)
Severn	Fluvial	524	306	160	100m	1.4M	4.8s (v)	22	2D + SG + ST	(Neal et al., 2015)
Global flood model*	Fluvial	4800	4800	23,040	0.00028°	345600	1.0s (f)	1119	2D + SG	(Sampson et al., 2015)
2D dry test	Synthetic	1288	4800	6162	4m	8000	0.1s (f)	0	2D	–
2D dry test with channels	Synthetic	1288	4800	6162	4m	8000	0.1s (f)	0	2D + SG	–
Pluvial test**	Pluvial	1850	1850	96	0.00083°	14943	1.7s (v)	96 k	2D + RR	(Sampson et al., 2015)
2D wet test	Synthetic	1288	4800	6162	4m	8000	0.1s (f)	0	2D	–
2D wet test with channels	Synthetic	1288	4800	6162	4m	8000	0.1s (f)	0	2D + SG	–

compiler. Note the hint to the compiler (#pragma simd) indicating that it should be possible to vectorise the numerical scheme.

We also tested two ways of implementing the most numerically intensive component of the computation where the hydraulic radius R is raised to the power of 4/3, by comparing the use of a generic c++ power function (POW(R, 4.0/3.0)) against multiplying R by itself four times and taking the cubed root of that (CBRT(R*R*R*R)).

3. Test cases

Hydraulic models are used for a wide variety of applications, meaning the performance of the modelling program needs to be robust across a representative range of test cases. In particular, computational performance is often found to change with the number of cells and the distribution of wet cells within the model domain (Neal et al., 2009b). Therefore, the new code was assessed using 12 models developed during previous research projects. These models are listed in Table 1, along within basic information on the processes simulated by each model and their size. The models also have different grid resolutions, time-steps and number of simulation time steps, which we include in Table 1 for completeness. The models also have different boundary conditions ranging from no boundary conditions (2D dry and 2D wet tests) to rainfall inputs into every cell (pluvial test). Most test cases have point inflow boundaries (Carlisle, EA test 5, Glasgow, Inner Niger Delta and Severn), with some having additional edge boundaries to allow flow to leave the domain (Carlisle, Inner Niger Delta, Severn). New York has a time varying water surface boundary. The total number of boundary cells is reported in Table 1. The main aim of the test case selection was to characterise the performance of the two-dimensional floodplain solver, hence most of the test cases are models of differing sizes that only use this solver. However, a key reason for using a CPU over a GPU architecture is the flexibility to add physical processes as additional modules, thus some models include additional physical processes (modules column in Table 1). Detailed descriptions of each model will not be provided here but can be found in the referenced sources where appropriate. However, to aid the discussion of the results the digital elevation models (Fig. 4), maximum simulated depths (Fig. 5) and percent of domain flooded over time (Fig. 6) are presented for the non-synthetic test cases (e.g. those with realistic topography and inundation patterns). The synthetic tests cases are not plotted because they all use a DEM of zero elevation everywhere and have a constant percentage of the domain flooded.

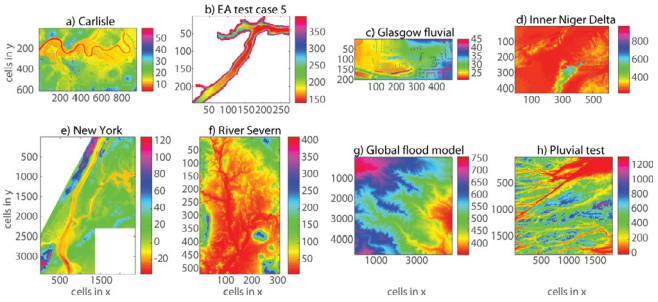

Fig. 4. Digital elevation models for eight test cases. All vertical units are in metres.

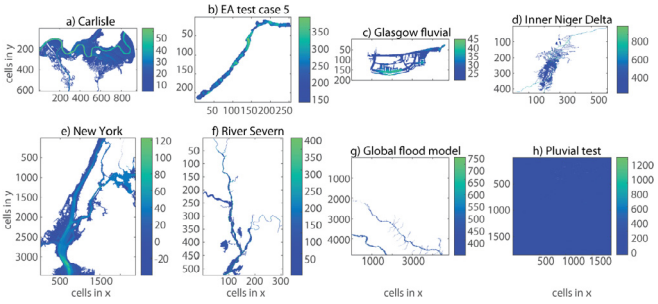

Fig. 5. Maximum simulated depth in metres.

4. Results

To assess the performance of the new code the 12 test cases were run on a dedicated node of the University of Bristol supercomputer BlueCrystal, which has 16 × 2.6 GHz Intel E5-2670 SandyBridge cores with 4GB/core of RAM. Therefore, simulations were run on up to 16 real cores, with cores left idle when less than 16 threads were created. For the 16 core simulations each model was run three times, with the shortest simulation time presented here. Other simulations were run just once due to the longer simulation times on fewer cores. In this paper, simulation time represents only the computation time needed to undertake the simulation and excludes the reading and writing of results at the start and end of the simulation as these depend on the supercomputer file store, which is shared by other users. The Intel C++ compiler version 13.1 for Linux was used throughout.

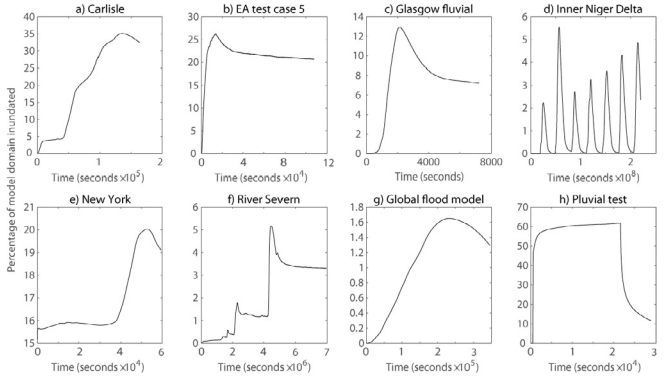

Fig. 6. Simulated inundation extent as a percentage of the model domain for each test case.

Table 2 records the compute times for the 12 test cases for the original LISFLOOD-FP model and optimised version on one and 16 cores. The global flood model simulation required the longest simulation time of up to ~275 k second (76 h), while the dry test case could take as little as 0.3 s. Parallel speedups for the two versions of the code are also shown (OMP speedup), along with a comparison between code on 1 and 16 cores.

Before considering the implications of these results we will deal with three caveats relating to results highlighted in italic and red. For the original model the pluvial simulation, which is the only model to include the runoff routing scheme of Sampson et al. (2013), obtained the worst speedup of 3.2x. This was unsurprising given that the routing component of this model was not implemented in parallel in the original code and means that the 15.7 times speedup between the 16 core optimised and original model is largely attributable to this improvement. The other two highlighted models are the New York and global flood model test cases. For these models, single core original LISFLOOD-FP model simulation time has been estimated from the two core and eight core original LISFLOOD-FP simulations respectively. This was necessary due to the long compute times needed by these models exceeding the supercomputer time limits. This means that the parallel speedups for the original model are likely to have been overestimated in these cases, as would the improvement in simulation time between the original and optimised codes. Results from the optimised model and comparison between the respective 16 core simulation times are unaffected.

The synthetic dry test case had the greatest speedup between the optimised and original code, with the optimised code executing two orders of magnitude faster. This was expected due to the implementation of wet edge tracking in the optimised version of the code. Parallel speedups for this test case with the optimised model are the lowest of any test case (3.5 x) due to the lack of work required by each thread. The synthetic all wet test case had the greatest parallel speedup for both the original and optimised codes (13.4 & 10.1 x respectively). Speedup between the original and optimised code was also relative high (6.7-10.1 x). Both these results were expected because the OMP threads will all have an equal amount of work to undertake, the vectorisation will be most efficient when all cells are wet and the computational work verses data accessed by the CPU will be maximised (e.g. you need to undertake the computationally expensive flow calculation for every cell interface in the domain).

Although the synthetic test cases are interesting, they are not representative of most real applications and therefore the majority of model simulations run by scientists and practitioners. The remaining simulations on actual DEMs show parallel speedups of between 4.2 x and 8.2 x, with large wet test cases such as New York tending to parallelise more efficiently than small domains such as Glasgow and dry domains such as EA test 5 (see max extents in Fig. 5 and percentage wet statistics in Fig. 6). That larger domains tend to improve parallel efficiency has been well reported in the literature (Leandro et al., 2014; Neal et al., 2009a). The presence of the 1D channel model generally reduces the parallel speedup. Interestingly, the speedup via code optimisation was greater than the speedup due to parallelisation in over half of the test cases (highlighted in Table 2 in bold) and of similar order in the others. Therefore, we find that optimising the code layout to allow for vectorisation, implementing a wet dry edge tracking approach and minimising the data movements during computation are as beneficial in terms of runtime as parallelisation on the processors used here.

It is difficult to explicitly quantify the benefits of each improvement we made to the code because there is likely to be strong interaction effects between the various changes. For example, the combined effect of reducing the data movement and implementing vectorisation will not be the sum of the two efforts in isolation. It was also not possible to implement the vectorisation without significantly changing the structure of the code and data from the original model. Nevertheless, we were able to disable a number of the optimisation steps to establish an indicative measure of how valuable these were in reducing the compute time. The results of disabling the vectorisation, wet edge tracking and numa alignment (a type of memory access optimisation) are summarised in Fig. 7, along with a comparison with the original model simulation times. All these results use floating point precision and 16 cores.

For the synthetic dry domain, the wet edge tracking is confirmed as the major contributor to the improvement in performance: disabling

Table 2
Simulation times in seconds for original code and optimised code on 1 and 16 cores. Speedup due to parallelisation is shown for each code (OMP speedup) and between the codes for the case of a single core run and 16 core run. Max speedup from a single core original model and 16 core optimised model are also shown.

Test case	Original model CPU time			Optimised model CPU time			Speedups between codes		
	1 core	16 cores	OMP speedup	1 core	16 cores	OMP speedup	1 core	16 cores	Max 1–16 speedup
Carlisle	9832.4	1639.2	6.0	1577.5	204.5	7.7	6.2	**8.0**	48.1
EA test 5	85.7	17.4	4.9	11.3	2.5	4.5	7.6	**7.0**	34.5
Glasgow fluvial	39.1	6.6	5.9	6.7	0.9	7.2	5.8	7.1	42.1
Inner Niger Delta	20104.1	2557.9	7.9	4530.9	403.0	11.2	4.4	6.3	49.9
New York	*58740*[a]	*7156.1*	*8.2*	7349.4	847.6	8.7	*8.0*	**8.4**	*69.3*
Severn	3826.7	683.7	5.6	916.8	112.7	8.1	4.2	6.1	34.0
Global model	*275190*^	*28545*	*9.6*	20106	4536.0	4.4	*13.7*	**6.3**	*60.7*
2D dry test	665.6	151.1	4.4	1.0	0.3	3.5	678.6	**539.0**	2374.9
2D dry + channels	518.5	73.9	7.0	1.0	0.3	3.5	525.8	**261.8**	1837.4
Pluvial test	7506.9	*2318.1*	*3.2*	1693.7	147.9	11.5	4.4	***15.7***	*50.8*
2D wet test	8904.5	809.9	11.0	811.7	80.0	10.1	11.0	10.1	111.3
2D wet + channels	7208.5	539.4	13.4	811.6	80.1	10.1	8.9	6.7	90.0

[a] Value estimated from 2 core simulation; ^ Value estimated for 8 core simulation.

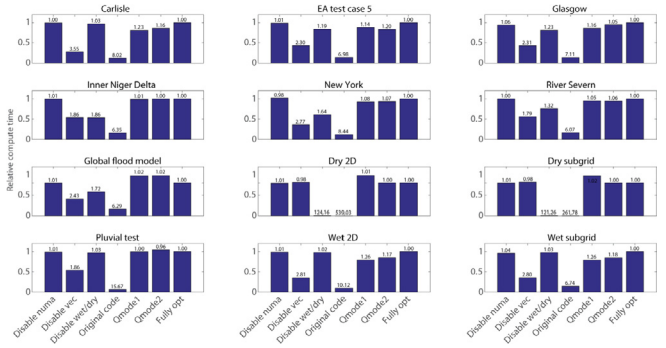

Fig. 7. Computation times for the 12 test cases relative to the fully optimised model (Fully opt). Plot shows comparison with the original code (Original code) and new code without specific features disabled. The disabled features in the fully optimised model are: the numa alignment of memory (Disable numa), solver vectorisation (Disable vec), the wet dry edge tracking (Disable wet dry). Also plotted are two methods for implementing the momentum equation (Qmode1 and Qmode2). All simulations were run on 16 cores.

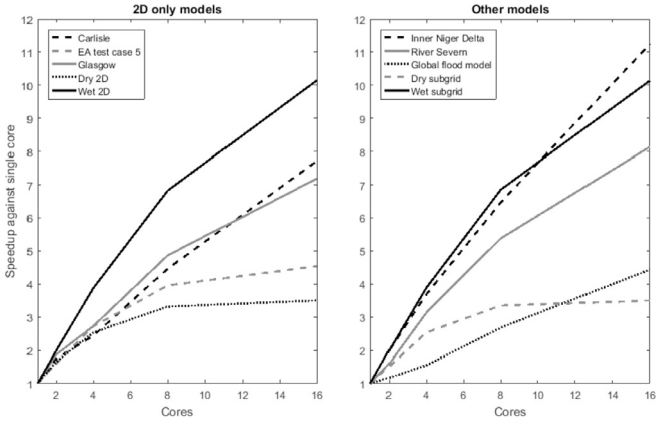

Fig. 8. Speedup of simulations for each model against number of cores. Models are split into those that only use the 2D model and those that have other components.

this feature increases the optimised model simulation time by 124 times (indicated by the number on top of the bars) for the 2D model. The vectorisation has essentially no effect, as expected due to the lack of any work to perform, therefore improvements to the code structure account for the remaining speedup over the original model (totalling 539 x). For the wet 2D simulation disabling the vectorisation increases the simulation time by 2.8 x. Since the single instruction multiple data vectorisation available on Sandy Bridge cores can accommodate up to four floating point numbers in parallel, this speedup is a substantial portion of the theoretical maximum 4 x.

For real test cases removing the vectorisation increased computation time between 1.8 x (Severn) and 3.5 x (Carlisle), with three models having values below 2 x (Inner Niger Delta, Severn, Pluvial test). The Inner Niger Delta and Severn domains are characterised by a relatively small and fragmented pattern of flooding connected by 1D channels (Figs. 5 and 6), while the runoff routing model in the pluvial test cases is not vectorised. These factors potentially explain the limited improvement brought about via vectorisation for these tests. However, as no test took longer when the vectorisation was enabled we conclude from these tests that the vectorisation will be universally beneficial in terms of compute time. Disabling the wet edge tracking resulted in a slowdown of between essentially nothing for Carlisle and 1.86 x for the seasonally dry Inner Niger Delta. As with vectorisation, the wet edge tracking was sufficiently cheap that no simulation showed a noticeable slowdown when it was implemented. Disabling numa alignment has no substantial effect on any test case, although this might become more significant on machines with more cores. It was impossible to isolate many of the reductions in data transfer between code functions that we made and these structural improvements to the code, as outlined by Fig. 1, are thought to yield most of the speedup from the original to optimised model not accounted for by vectorisation and wet edge tracking (e.g. in the region of 2-4 x for the real test cases).

For completeness we also include the model speedups by number of cores in Fig. 8 to examine the parallel efficiency of the code. Parallel efficiency varied strongly with the distribution of wet cells in the domain. The completely wet models remain near 100% efficient in terms of speedup until 8 cores are in use. However, the dry test shows essentially no speedup after 4 cores, due to the lack of parallel computation relative to serial overheads. Except for the dry test cases, parallel efficiency remained similar between 4 and 16 cores suggesting the parallelisation is scaling well.

5. Comment on results relative to previous studies

Benchmarking of hydrodynamic models from a computational performance perspective is a difficult task because the test case can have a significant influence on the relative model performance. The solvers used and their accuracies vary substantially and codes are rarely compared on identical hardware meaning relative performance might vary by hardware and compiler. Nevertheless, it is worth placing the results here in a wider context. The benchmarking exercise by Néelz and Pender (2010) was one of the most comprehensive efforts to benchmark the main commercial and research focused two-dimensional hydrodynamic modelling codes. They report simulation times for several models for EA test 5 and the hardware used for the simulation. Although the Néelz and Pender study is a few years old, it did report results for the original LISFLOOD-FP model which allows a comparison to be made. Simulation times of 9–168 s were reported by Néelz and Pender for EA test 5 and are reproduced in Table 3. The original LISFLOOD-FP model required 28.2 s using 8 cores and was competitive on simulation time but not especially quick for this particular case (note that the relative position of the models varied from test to test in Néelz and Pender (2010)). In this study, the same simulation on an identical number of cores was 1.3 x faster at 21.5 s for the original model, reducing to 2.8 s (~10 x) for the optimised model on eight cores. Although this comparison is rather limited for the reasons outlined above it confirms that our optimisation efforts are relevant and substantive within the wider flood inundation modelling context.

6. Conclusions

For 2D hydrodynamic simulations our code developments yielded between 4.2 and 8.4 x speedups when the model was run on the same number of cores, while the speedups from a single core implementation of the original LISFLOOD-FP to 16 core simulations on the new code was between 34 and 60 x. Under idealised conditions where the whole domain was wet code speedup was up to 111 x. Interestingly, roughly the same improvement in numerical efficiency was achieved through code development as was achieved through parallelisation on a 16 core CPU. In relation to the four development areas identified in this paper the following conclusions can be drawn from this work:

6.1. Parallel loop structure

For shared memory parallelisation using OpenMP the original version of LISFLOOD-FP yielded speedups of ~5-13 x on 16 cores by simply implementing *for* loops in parallel. By restructuring the parallelisation to create threads at the start of the simulation rather than within each loop it was possible to maintain this parallel efficiency despite other developments to the code reducing the work done by the rest of the model by a similar margin. This change is likely to be

Table 3
Simulation times for EA test 5 from Néelz and Pender (2010).

Code	Simulation time (s)	Cores used	Clock speed (GHz)	Processor	Numerical scheme[a]
Results reported by Néelz and Pender (2010)					
Flowroute	9	4	4.0	Intel Core i7-950	FD explicit
Infoworks	9	12	2.4	2 x Intel Xeon E5645	FV explicit unstructured
Flood Modeller	58.5	1	2.8	Intel Xeon W3530	FD implicit
JFLOW +	22	285	1.47	NVIDIA GeForce GTX	FV explicit
LISFLOOD-FP original	28.2	8	2.8	Intel Xeon E5440	FD explicit
MIKE FLOOD	28.3	8	2.2	Intel Core i7- 2670QM	FD implicit
SOBEK	168	1	2.66	Intel i7	FD implicit
TUFLOW	26	1	3.4	Intel Core i7-2600	FD implicit
This paper					
LISFLOOD-FP original	85.7	1	2.6	Intel Xeon E5-2670	FD explicit
LISFLOOD-FP original	21.5	8	2.6	Intel Xeon E5-2670	FD explicit
LISFLOOD-FP original	17.4	16	2.6	Intel Xeon E5-2670	FD explicit
LISFLOOD-FP opt	11.3	1	2.6	Intel Xeon E5-2670	FD explicit
LISFLOOD-FP opt	2.8	8	2.6	Intel Xeon E5-2670	FD explicit
LISFLOOD-FP opt	2.5	16	2.6	Intel Xeon E5-2670	FD explicit

[a] FD is finite difference and FV finite volume. All models used Cartesian grids unless specified as unstructured.

applicable to many models where processes are often written as separate sub-routines with their own loops and parallel loop definitions. The disadvantage of the restructuring was that care had to be taken to identify areas of the code that could not run efficiently in parallel (for example structures) or that must run sequentially (for example calculations of momentum and continuity). However, these cases could all be handled with barrier and omp single commands that force all threads to complete before moving on and force a single thread implementation, respectively.

6.2. Wet and dry edge tracking

Implementing a simple wet edge tracking approach yielded substantial improvements in compute time for most realistic tests cases, while the overhead of tracking the edge was sufficiently low that this was not observed in the simulation where the whole domain was wet. Implementing such a scheme, where something more advanced doesn't already exist, would therefore be expected to yield a 0-2 x speedup for most test cases and substantially more for very dry domains.

6.3. Data access

Improving memory access by the code was believed to be a substantial limiting factor on numerical efficiency. In the tests conducted here it is difficult to isolate how the restructuring of the code improved compute time. However, given the overall model performance statistics and the performance of the models when other optimisations were disabled it is likely that these developments account for a halving of the compute time over the original simulation time (2 x speedup). Aligning data access and improving the data movement during simulations was also an intrinsic component of the vectorisation process and we suspect this will not have been so successful without this initial reorganisation of the code structure.

The most expensive component of the momentum equation is raising the hydraulic radius to the power of 4/3. We tested the use of a specific cubed root function over the more general power function and found a small improvement in model performance on some test cases.

6.4. Vectorisation

The benefits of vectorising the solver will vary depending on the CPU, however for an SIMD register that supports a vector length of four floating point number speedups was as high as 3.5 x when AVX was enabled. Furthermore, enabling vectorisation did not increase computation time for any of our tests cases even when the flood inundation was quite fragmented (e.g. Inner Niger Delta).

Overall this paper has documented several model development steps that can yield quite substantial improvements in model performance on standard computer hardware. These improvements were more substantial than the reduction in computation time of 3-5 x between a full shallow water model and the local inertia implementation, when both were implemented in the LISFLOOD-FP code (Neal et al., 2012b). We hope that other developers and researchers will find the steps we have taken a useful when considering their own model development plans. There are several numerical schemes that directly use the numerical approach adopted here (e.g. Adams et al., 2017a; b; Coulthard et al., 2013; Courty et al., 2017; Dottori et al., 2016), however all explicit hydrodynamic model on regular grids and many process based models in geosciences have a small stencil of neighbours where information cannot travel more than one cell in any time-step. These could therefore all be optimised with the methods outlined here.

7. Software and data availability

The LISFLOOD-FP software is developed by the University of Bristol. A freeware version of the code for non-commercial use can be downloaded from the universities website http://www.bristol.ac.uk/geography/research/hydrology/models/lisflood/downloads/. The Lead developers are Dr Jeffrey Neal (corresponding author) and Prof. Paul Bates at the School of Geographical Sciences, University of Bristol, BS8 1SS, UK. LISFLOOD-FP is written in C + + and can be complied for Windows and Linux. Version 6 of the code, used in this study, requires a processor hardware with AVX capability or newer. The code is not open source, however we give access to the LISFLOOD-FP code repository to numerous research collaborators. Researchers interested in accessing the code are encouraged to email the lead authors or access the open source version of the code associated with the paper by Hoch et al. (2017). The test cases from this paper that use open data are available on the Mendeley link.

Acknowledgements

Toby Dunne was supported by the NERC Impact Accelerator Account at the University of Bristol, while Chris Sampson and Andy Smith were supported by NERC via NE/M007766/1.

Appendix A

```
#pragma ivdep
#pragma simd // note this pragma is here as a hint to the compiler that this should be vectorized
for (int i = row_start_x; i < row_end_x; i++)
{
  int index = grid_row_index + i; // next column
  int index_next = index + 1; // next column

  NUMERIC_TYPE h0 = h_grid[index]; // water depth in cell i
  NUMERIC_TYPE h1 = h_grid[index_next]; // water depth in cell i+1
  NUMERIC_TYPE z0 = dem_grid[index]; // DEM elevation in cell i
  NUMERIC_TYPE z1 = dem_grid[index_next]; // DEM elevation in cell i+1
  NUMERIC_TYPE surface_elevation0 = z0 + h0; // water surface elevation in cell i
  NUMERIC_TYPE surface_elevation1 = z1 + h1; // water surface elevation in cell i+1
  // Calculating depth of flow (hflow) based on floodplain levels
  NUMERIC_TYPE hflow = getmax(surface_elevation0, surface_elevation1)- getmax(z0, z1);
  NUMERIC_TYPE q_tmp, surface_slope;

  if (hflow > depth_thresh) // if cell depths is above a minimum threshold (default 0.001 m)
  {
    NUMERIC_TYPE area = (row_dy)* hflow; // flow cross-sectional area
    NUMERIC_TYPE dh = (surface_elevation0)-(surface_elevation1); // Change in water surface elevation
    surface_slope = -dh / row_dx; // water surface slope
    q_tmp = CalculateQ(surface_slope, hflow, delta_time, g, area, g_friction_sq_x_grid[index_next],
Qx_old_grid[index_next]); // calculate flows between cells (eq. 1)
  }
  else q_tmp = C(0.0); // if cell is dry ensure flow is set to zero
  Qx_old_grid[index_next] = q_tmp;
  }
int count = row_end_x - row_start_x;
if (count > 0) memcpy(Qx_grid + grid_row_index + row_start_x + 1, Qx_old_grid + grid_row_index +
row_start_x + 1, sizeof(NUMERIC_TYPE) * count);

/// Calculate channel flow using inertial wave equation ///
inline NUMERIC_TYPE CalculateQ(const NUMERIC_TYPE surface_slope,
        NUMERIC_TYPE R, // hydraulic radius
        const NUMERIC_TYPE delta_time, // time step
        const NUMERIC_TYPE g, // gravity
        const NUMERIC_TYPE area, //flow area (not cell area)
        const NUMERIC_TYPE g_friction_squared, // gravity*(manning's n^2)
        const NUMERIC_TYPE q_old) // flow from previous time step
{
```

```
        // calculate flow based on m^3 formula (note power is 4/3, profiling shows performance gain by
multiply and cuberoot)
#if _CALCULATE_Q_MODE == 0
        NUMERIC_TYPE pow_tmp1, pow_tmp, abs_q, calc_num, calc_den;

        pow_tmp1 = R * R * R * R;
        pow_tmp = CBRT(pow_tmp1); // 4 multiplies and 1 cube root profiled faster than POW(R,4/3)

        abs_q = FABS(q_old);

        calc_num = (q_old - g * area * delta_time * surface_slope);
        calc_den = (1 + delta_time * g_friction_squared * abs_q / (pow_tmp * area));
        return calc_num / calc_den;
#else
#if _CALCULATE_Q_MODE == 1
        NUMERIC_TYPE pow_tmp1, pow_tmp, abs_q, calc_num, calc_den;

        pow_tmp = POW(R, C(4.0)/C(3.0));

        abs_q = FABS(q_old);

        calc_num = (q_old - g * area * delta_time * surface_slope);
        calc_den = (1 + delta_time * g_friction_squared * abs_q / (pow_tmp * area));
        return calc_num / calc_den;
#else

        return (q_old - g * area * delta_time * surface_slope) / ((1 + delta_time * g_friction_squared *
FABS(q_old) / (POW(R, C(4.0)/C(3.0)) * area)));
#endif
#endif
```

Appendix A: Code snippets describing an implementation of the LISFLOOD-FP momentum equation that can be vectorised using advanced vector extension. Where *i* is the cell index, *h* is water depth, *z* is bed elevation, *hflow* cross sectional depth of flow, *area* is the cross-sectional flow area, *dh* is the difference in water surface elevation between adjacent cells, *row_dy* and *row_dx* are the cell widths in x and y directions, *delta_time* is the model time step, *g* is acceleration due to gravity, *g_friction_sq_x_grid* is the friction squared and *Qx_old_grid* is the discharge from the previous time step.

References

Adams, J.M., Gasparini, N.M., Hobley, D.E.J., Tucker, G.E., Hutton, E.W.H., Nudurupati, S.S., Istanbulluoglu, E., 2017a. The Landlab v1.0 OverlandFlow component: a Python tool for computing shallow-water flow across watersheds. Geosci. Model Dev. 10 (4), 1645–1663.

Adams, J.M., Gasparini, N.M., Hobley, D.E.J., Tucker, G.E., Hutton, E.W.H., Nudurupati, S.S., Istanbulluoglu, E., 2017b. The Landlab v1.0 OverlandFlow component: a Python tool for computing shallow-water flow across watersheds. Geosci. Model Dev. 10 (4), 1645–1663. http://dx.doi.org/10.5194/gmd-10-1645-2017.

Alfieri, L., Salamon, P., Bianchi, A., Neal, J., Bates, P., Feyen, L., 2014. Advances in pan-European flood hazard mapping. Hydrol. Process. 28 (13), 4067–4077. http://dx.doi.org/10.1002/hyp.9947.

Aronica, G.T., Franza, F., Bates, P.D., Neal, J.C., 2012. Probabilistic evaluation of flood hazard in urban areas using Monte Carlo simulation. Hydrol. Process. 26 (26), 3962–3972. http://dx.doi.org/10.1002/Hyp.8370.

Barkwith, A., Hurst, M.D., Jackson, C.R., Wang, L., Ellis, M.A., Coulthard, T.J., 2015. Simulating the influences of groundwater on regional geomorphology using a distributed, dynamic, landscape evolution modelling platform. Environ. Model. Software 74, 1–20.

Bates, P., Trigg, M., Neal, J.C., Dabrowa, A., 2016. LISFLOOD-fp User Manual, Edited. University of Bristol, Bristol, UK.

Bates, P.D., Horritt, M.S., Fewtrell, T.J., 2010. A simple inertial formulation of the shallow water equations for efficient two-dimensional flood inundation modelling. J. Hydrol. 387 (1–2), 33–45. http://dx.doi.org/10.1016/j.jhydrol.2010.03.027.

Bradbrook, K., 2006. JFLOW: a multiscale two-dimensional dynamic flood model. Water Environ. J. 20 (2), 79–86. http://dx.doi.org/10.1111/j.1747-6593.2005.00011.x.

Coulthard, T.J., Neal, J.C., Bates, P.D., Ramirez, J., de Almeida, G.A.M., Hancock, G.R., 2013. Integrating the LISFLOOD-FP 2D hydrodynamic model with the CAESAR model: implications for modelling landscape evolution. Earth Surf. Process. Landforms 38 (15), 1897–1906.

Courty, L.G., Pedrozo-Acuña, A., Bates, P.D., 2017. Itzï (version 17.1): an open-source, distributed GIS model for dynamic flood simulation. Geosci. Model Dev. 10 (4), 1835–1847. http://dx.doi.org/10.5194/gmd-10-1835-2017.

de Almeida, G.A.M., Bates, P., Freer, J.E., Souvignet, M., 2012. Improving the stability of a simple formulation of the shallow water equations for 2-D flood modeling. Water Resour. Res. 48 (5). http://dx.doi.org/10.1029/2011wr011570. W05528.

de Moel, H., van Alphen, J., Aerts, J., 2009. Flood maps in Europe - methods, availability and use. Nat. Hazards Earth Syst. Sci. 9 (2), 289–301.

de Paiva, R.C.D., Buarque, D.C., Collischonn, W., Bonnet, M.-P., Frappart, F., Calmant, S., Bulhões Mendes, C.A., 2013. Large-scale hydrologic and hydrodynamic modeling of the Amazon River basin. Water Resour. Res. 49 (3), 1226–1243. http://dx.doi.org/10.1002/wrcr.20067.

Dottori, F., Salamon, P., Bianchi, A., Alfieri, L., Hirpa, F.A., Feyen, L., 2016. Development and evaluation of a framework for global flood hazard mapping. Adv. Water Resour. 94, 87–102. http://dx.doi.org/10.1016/j.advwatres.2016.05.002.

Duckworth, M., 2014. Exposure of the New York Metropolitan Area to Sea Level Rise: Hurricane Sandy Case Study. Undergraduate Dissertation. School of Geographical Sciences University of Bristol, UK.

Fewtrell, T.J., Bates, P.D., Horritt, M., Hunter, N.M., 2008. Evaluating the effect of scale in flood inundation modelling in urban environments. Hydrol. Process. 22 (26), 5107–5118. http://dx.doi.org/10.1002/Hyp.7148.

Gibson, M., 2015. Genetic Programming and Cellular Automata for Fast Flood Modelling on Multi-core CPU and Many-core GPU Computers. University of Exeter, Exeter.

Gouldby, B., Sayers, P., Mulet-Marti, J., Hassan, M.A.A.M., Benwell, D., 2008. A methodology for regional-scale flood risk assessment. ICE - Water Man 161, 169–182.

Hoch, J.M., Neal, J.C., Baart, F., van Beek, R., Winsemius, H.C., Bates, P.D., Bierkens, M.F.P., 2017. GLOFRIM v1.0-A globally applicable computational framework for integrated hydrological-hydrodynamic modelling. Geosci. Model Dev. 10 (10), 3913–3929.

Hunter, N.M., et al., 2008. Benchmarking 2D hydraulic models for urban flooding. Proc. Inst. Civ. Eng.-Water Manag. 161 (1), 13–30. http://dx.doi.org/10.1680/wama.2008.161.1.13.

Judi, D., Burian, S., McPherson, T., 2011. Two-dimensional fast-response flood modeling: desktop parallel computing and domain tracking. J. Comput. Civ. Eng. 25 (3), 184–191. http://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000064.

Kalyanapu, A.J., Shankar, S., Pardyjak, E.R., Judi, D.R., Burian, S.J., 2011. Assessment of GPU computational enhancement to a 2D flood model. Environ. Model. Software 26 (8), 1009–1016. http://dx.doi.org/10.1016/j.envsoft.2011.02.014.

Lamb, R., Crossley, M., Waller, S., 2009. A fast two-dimensional floodplain inundation model. Proc. Inst. Civ. Eng.-Water Manag. 162 (6), 363–370. http://dx.doi.org/10.1680/wama.2009.162.6.363.

Leandro, J., Chen, A.S., Schumann, A., 2014. A 2D parallel diffusive wave model for floodplain inundation with variable time step (P-DWave). J. Hydrol. 517, 250–259. http://dx.doi.org/10.1016/j.jhydrol.2014.05.020.

Neal, J., Fewtrell, T., Trigg, M., 2009a. Parallelisation of storage cell flood models using OpenMP. Environ. Model. Software 24 (7), 872–877. http://dx.doi.org/10.1016/j.envsoft.2008.12.004.

Neal, J., Fewtrell, T.J., Trigg, M.A., 2009b. Parallelisation of storage cell flood models using OpenMP. Environ. Model. Software 24 (7), 872–877.

Neal, J., Schumann, G., Bates, P., 2012a. A subgrid channel model for simulating river

hydraulics and floodplain inundation over large and data sparse areas. Water Resour. Res. 48 (11), W11506,. http://dx.doi.org/10.1029/2012wr012514.

Neal, J., Keef, C., Bates, P., Beven, K., Leedal, D., 2013. Probabilistic flood risk mapping including spatial dependence. Hydrol. Process. 27 (9), 1349–1363. http://dx.doi.org/10.1002/hyp.9572.

Neal, J.C., Odoni, N.A., Trigg, M.A., Freer, J.E., GarciaPintado, J., Mason, D., Wood, M., Bates, P.D., 2015. Efficient incorporation of channel cross-section geometry uncertainty into regional and global scale flood inundation models. J. Hydrol. 529, 169–183. http://dx.doi.org/10.1016/j.jhydrol.2015.07.026. ISSN 0022-1694. Available at: http://centaur.reading.ac.uk/40846/.

Neal, J., Schumann, G., Fewtrell, T., Budimir, M., Bates, P., Mason, D., 2011. Evaluating a new LISFLOOD-FP formulation with data from the summer 2007 floods in Tewkesbury, UK. J. Flood Risk Manag. 4 (2), 88–95. http://dx.doi.org/10.1111/j.1753-318X.2011.01093.x.

Neal, J., Villanueva, I., Wright, N., Willis, T., Fewtrell, T., Bates, P., 2012b. How much physical complexity is needed to model flood inundation? Hydrol. Process. 26 (15), 2264–2282. http://dx.doi.org/10.1002/Hyp.8339.

Neal, J.C., Fewtrell, T.J., Bates, P.D., Wright, N.G., 2010. A comparison of three parallelisation methods for 2D flood inundation models. Environ. Model. Software 25 (4), 398–411. http://dx.doi.org/10.1016/j.envsoft.2009.11.007.

Neal, J.C., Bates, P.D., Fewtrell, T.J., Hunter, N.M., Wilson, M.D., Horritt, M.S., 2009. Distributed whole city water level measurements from the Carlisle 2005 urban flood event and comparison with hydraulic model simulations. J. Hydrol. 368 (1–4), 42–55. http://dx.doi.org/10.1016/j.jhydrol.2009.01.026.

Néelz, S., Pender, G., 2010. Benchmarking of 2D Hydraulic Modelling Packages.*Rep.* Environment Agency, Bristol.

Petaccia, G., Leporati, F., Torti, E., 2016. OpenMP and CUDA simulations of Sella Zerbino Dam break on unstructured grids. Comput. Geosci. 1–10. http://dx.doi.org/10.1007/s10596-016-9580-5.

Sampson, C.C., Bates, P.D., Neal, J.C., Horritt, M.S., 2013. An automated routing methodology to enable direct rainfall in high resolution shallow water models. Hydrol. Process. 27 (3), 467–476. http://dx.doi.org/10.1002/hyp.9515.

Sampson, C.C., Smith, A.M., Bates, P.B., Neal, J.C., Alfieri, L., Freer, J.E., 2015. A high-resolution global flood hazard model. Water Resour. Res. 51 (9), 7358–7381. http://dx.doi.org/10.1002/2015wr016954.

Sanders, B.F., Schubert, J.E., Detwiler, R.L., 2010. ParBreZo: a parallel, unstructured grid, Godunov-type, shallow-water code for high-resolution flood inundation modeling at the regional scale. Adv. Water Resour. 33 (12), 1456–1467. http://dx.doi.org/10.1016/j.advwatres.2010.07.007.

Schubert, J.E., Sanders, B.F., 2012. Building treatments for urban flood inundation models and implications for predictive skill and modeling efficiency. Adv. Water Resour. 41, 49–64. http://dx.doi.org/10.1016/j.advwatres.2012.02.012.

Vacondio, R., Dal Palù, A., Ferrari, A., Mignosa, P., Aureli, F., Dazzi, Susanna, 2017. A non-uniform efficient grid type for GPU-parallel Shallow water equations models. Environ. Modell. Soft. 88, 119–137.

Vousdoukas, M.I., Voukouvalas, E., Mentaschi, L., Dottori, F., Giardino, A., Bouziotas, D., Bianchi, A., Salamon, P., Feyen, L., 2016. Developments in large-scale coastal flood hazard mapping. Nat. Hazards Earth Syst. Sci. 16 (8), 1841–1853. http://dx.doi.org/10.5194/nhess-16-1841-2016.

Winsemius, H.C., Van Beek, L.P.H., Jongman, B., Ward, P.J., Bouwman, A., 2013. A framework for global river flood risk assessments. Hydrol. Earth Syst. Sci. 17 (5), 1871–1892. http://dx.doi.org/10.5194/hess-17-1871-2013.

Yamazaki, D., Kanae, S., Kim, H., Oki, T., 2011. A physically based description of floodplain inundation dynamics in a global river routing model. Water Resour. Res. 47 doi: W0450110. 1029/2010wr009726.

Yu, D., Lane, S.N., 2006. Urban fluvial flood modelling using a two-dimensional diffusion-wave treatment, part 1: mesh resolution effects. Hydrol. Process. 20 (7), 1541–1565.