Parallelization of a two-dimensional flood inundation model based on domain decomposition

Dapeng Yu*

Department of Geography, Loughborough University, Leicester, UK LE11 3TU.

Abstract

Flood modelling often involves prediction of the inundated extent over large spatial and temporal scales. As the dimensionality of the system and the complexity of the problems increase, the need to obtain quick solutions becomes a priority. However, for large-scale problems or situations where fine resolution data is required, it is often not possible or practical to run the model on a single computer in a reasonable timeframe. This paper presents the development and testing of a parallelized 2D diffusion-based flood inundation model (FloodMap-Parallel) which enables large-scale simulations to be run on distributed multi-processors. The model has been applied to three locations in the UK with different flow and topographical boundary conditions. The accuracy of the parallelized model and its computational efficiency have been tested. The predictions obtained from the parallelized model match those obtained from the serialized simulations. The computational performance of the model has been investigated in relation to the granularity of the domain decomposition, the total number of cells and the domain decomposition configuration pattern. Results show that the parallelized model is more effective with simulations of low granularity and a large number of cells. The large communication overhead associated with the potential load-imbalance between sub-domains is a major bottleneck in utilizing this approach with higher domain granularity.

Key words: flood inundation modelling, diffusion-wave, parallelization, domain decomposition, MPI, HPC.

Correspondence to:
D Yu, *Department of Geography, Loughborough University, Leicester, UK LE11 3TU.*
Tel: 0044 (0)1509 22 8191, Email: d.yu2@lboro.ac.uk

1. Introduction

Simplified raster-based 2D models (Bates and De Roo, 2000; Bradbrook *et al.*, 2004; Yu and Lane 2006a, b) are particularly suitable for flood inundation modelling because they are relatively easy to integrate with topographic data and possess good computational efficiency compared to models that solve the full solution of 2D shallow water equations. Recent developments in data capture techniques (for example, LiDAR) have provided high-resolution and high-accuracy data, initiating a rapid shift from a data-poor to a data-rich and spatially complex modelling environment (Bates *et al.*, 2003). Indeed, for most applications, topographic data availability is no longer a limiting factor. With high-resolution numerical meshes, it is possible to represent explicitly the impacts of topographical complexity on flow routing, provided that the numerical schemes are stable, not prone to numerical diffusion, and computationally feasible. Despite the increasing availability of high-resolution topographic data, only limited spatial areas have been considered to date (Fewtrell *et al.*, 2008) and applications have rarely utilised topographic resolutions that are commensurate with feature dimensions (e.g. Neal *et al.*, 2009). For example, Mark *et al.* (2004) suggested that models of urban flooding must achieve a resolution (1–5 m); that is, high enough to resolve individual structural features. However, computational constraints mean that it would be challenging to achieve this type of resolution in urban settings, especially over large spatial and temporal scales. In such situations, finer topographic data are typically re-sampled to coarser meshes. Raster-based flood inundation model was found to be very sensitive to mesh resolution and coarsening of mesh resolution can have a significant impact on the accuracy of model prediction (e.g. Yu and Lane 2006a; Fewtrell, *et al.*, 2008). The under-utilisation of high-resolution topographic data is due, on one hand, to the exceptionally high computational requirements associated with fine-scale grids; and on the other, to the small time steps required by this type of model in order to achieve computational stability (e.g. Hunter *et al.*, 2005). Hunter *et al.* (2005) developed an adaptive time-stepping solution for situations where the dynamic wave propagation needs to be simulated correctly. This approach incurs further computational costs as the time step shows an approximate quadratic and linear dependence on grid cell size and friction coefficient (Hunter *et al*., 2005). There are different ways to improve this situation, e.g. with more efficient codes, new numerical schemes, or through improvement to the current approach. One example is the sub-grid treatment approach developed by Yu and Lane (2006b). This takes into account the variability of sub-grid topography in a coarser mesh to decrease computational requirements. Taking advantages of the development in computing technology, parallelization is another way to improve the applicability of flood inundation modelling over large spatial and temporal domains. There are in general two ways to implement parallel computing: data decomposition and functional decomposition. Functional decomposition can be achieved by having nodes execute different portions of the same code and each node performs a specific sub-task simultaneously. Data decomposition is achieved by distributing the computational domain of a problem among nodes and each node is given a subset of the data to process. To date, the dominant approach to the parallelization of numerical models such as flood inundation has been data decomposition. Data decomposition can be implemented through either explicit domain decomposition (DD) coupled with Message Passing Interface (MPI) (e.g. Hervouet, 2000; Rao, 2005); or implicit data decomposition through shared-memory parallelization (e.g. OpenMP). The DD approach based on MPI normally divides the problem domain into *n*-processor blocks of fixed size whilst the shared memory approach distributes individual or groups of rows of the domain between threads. For example, the parallelization of a 2D finite-element hydrodynamic model TELEMAC-2D was implemented by Hervouet (2000) using message passing software PVM (Parallel Virtual Machine) based on domain decomposition. This investigation has been focusing on the speedups and efficiencies of simulations on machines with different specifications and operation systems. Up to 8 processors were utilized and the best speedup achieved was 6.2 with an 8-processor implementation. Another example of hydrodynamic model parallelization using domain decomposition was provided by Rao (2005). The finite-element code RMA2 was parallelized under portable extensible toolkit for scientific computation (PETSc) environment which is based on MPI protocols. A sample test of a relatively small size (200 elements) was carried out on machines with 1, 2, 4 and 8 processors. In this case, near-theory speedups were achieved in all cases. More recently, Neal *et al.* (2009) reported a parallelized version of a flood inundation model (LISFLOOD-FP, Bates and De Roo, 2000) implemented with shared-memory OpenMP. A range of simulations of different problem size was carried out on machines with 4 or 8 processors. Given the same number of nodes, larger speedups were generally found to be associated with larger problem size. The serial time and load imbalances were thought to be the limiting factors for parallel speedup in this investigation. Shared-memory parallelization is usually easier to implement when compared with domain decomposition since it allows a serial code to be parallelized by inserting predefined complier directives. Apart from the fact that the number of shared memory processors in computers commercially available is generally smaller than 8, the major disadvantage of using shared-memory parallelization is associated with the rapid performance saturation as the number of CPUs increases.

Domain decomposition (DD) is regarded as a natural way to enable parallelization as the algorithms in a serial code can be carried over to the parallel case, with the addition of codes to deal with cross-boundary communications (Keyes, 1989). Raster-based 2D flood inundation models are favourable types of models for parallel computing as they are relatively easy for domain decomposition due to the gridded-environment. Neal *et al*. (2010) reported a comparison study of three parallelization methods for a 2D flood inundation model (LISFLOOD-FP) including OpenMP, MPI and specialized accelerator cards. The DD approach implemented with MPI was found to offer good speedups and efficiencies, in particular in test cases where the computational domain is fully wet. This study presents the development and initial testing of a parallelized flood inundation model through domain decomposition. The focus of this study is on the exploration of parallelization speedups and efficiencies in relation to the size of problem domain and the DD pattern in the context of dynamic flood inundation modelling. Section 2 presents the model structure and how the parallelism is implemented. Sections 3 and 4 discuss the case studies of the parallelization and the results obtained in terms of the model predictive and computational performance. Section 5 concludes and discusses the directions for future development of this approach.

2. Model Description and Parallel Implementation

FloodMap (Yu and Lane 2006a, b) is a two-dimensional diffusion-based flood inundation model designed for fluvial flood inundation prediction in topographically complex floodplains. It has a similar structure to that of LISFLOOD-FP (Bates and De Roo, 2000) and JFLOW (Bradbrook *et al.*, 2004), but coded in Java. The model is the basis of the sub-grid treatment approach developed by Yu and Lane (2006b) and has been tested and verified with a range of boundary conditions and in a number of environments (Yu, 2005; Tayefi *et al.*, 2007; Lane *et al.*, 2007; Lane *et al.*, 2008). Yu and Lane (2006a) have reviewed the basis of the model that is modified here for parallelization, and so only the major model structure is outlined here.

Flux in a diffusion-based flood inundation model can be determined using Manning's equation whilst ignoring the inertial and advection forces of the flow. Manning's equation expressed for parameters in a regular computational mesh, takes the form:

$$Q = \frac{wd^{5/3}S^{0.5}}{n} \tag{1}$$

where Q is the flux out of a grid cell, *w* is the mesh resolution, *d* is the effective flow depth, *S* is the energy slope and *n* is Manning's roughness coefficient. Considering a regular grid cell and the four cells immediately adjacent to it, the orthogonal directions of the grid cells are termed *i* and *j*. Two parameters need to be derived from the configuration: the energy slope *S*, and the effective flow depth *d* to solve Equation (1). The energy slope in each orthogonal direction is given by the difference in water levels between the cells divided by the distance between the cell centres (2a, 2b).

Water is allowed to flow out of the cell only if the slope of the source cell to the other adjacent cells is positive.

$$S_i = \frac{h_{i,j} - h_{i\pm1,j}}{w} \tag{2a}$$

$$S_j = \frac{h_{i,j} - h_{i,j\pm1}}{w} \tag{2b}$$

The flow direction is determined by the vector sum of the energy slope in the *i* and *j* directions. Outflow is only allowed in two of the adjacent orthogonal directions defined by the vector sum of the slopes which is given by:

$$S = \sqrt{S_i^2 + S_j^2} \tag{3}$$

The effective flow depth in each of the four directions is determined as the water level in the source above the higher of the two ground levels along either the *i* or *j* direction as given by (4a, 4b).

$$d_i = h_{i,j} - \max\lfloor g_{i,j}, g_{i\pm1,j} \rfloor \tag{4a}$$

$$d_j = h_{i,j} - \max\lfloor g_{i,j}, g_{i,j\pm1} \rfloor \tag{4b}$$

where *d* is the effective flow depth, *h* is the water surface elevation and *g* is the ground elevation. The effective depth in the outflow direction is then calculated as the arithmetic mean of the two flow effective flow depths along the line of steepest slope:

$$d = \frac{d_i S_i^2 + d_j S_j^2}{S^2} \tag{5}$$

Substituting (3) and (5) into equation (1) solves Manning's equation partitioned on a regular grid. The flow vector can then be solved in each of the two orthogonal directions of the grid, giving possible flow in up to 2 of the adjacent cells at each time step. For each time step, the fluxes into and out of each cell in the calculation domain are then given by (6a) and (6b).

$$Q_i = Q\frac{S_i}{S} = \frac{wd^{5/3}S_i}{nS^{0.5}} = \frac{wd^{5/3}\left(\dfrac{h_{i,j} - h_{i\pm1,j}}{w}\right)}{n\left[\left(\dfrac{h_{i,j} - h_{i\pm1,j}}{w}\right)^2 + \left(\dfrac{h_{i,j} - h_{i,j\pm1}}{w}\right)^2\right]^{1/4}} \tag{6a}$$

$$Q_j = Q\frac{S_j}{S} = \frac{wd^{5/3}S_j}{nS^{0.5}} = \frac{wd^{5/3}\left(\dfrac{h_{i,j} - h_{i,j\pm1}}{w}\right)}{n\left[\left(\dfrac{h_{i,j} - h_{i\pm1,j}}{w}\right)^2 + \left(\dfrac{h_{i,j} - h_{i,j\pm1}}{w}\right)^2\right]^{1/4}} \tag{6b}$$

The parallelization was achieved through modification of the serial code and the setup of a server node. The communication between distributed nodes was managed by the server and no specific MPI software was used. The parallelized version of FloodMap (FloodMap-Parallel) distributes the data based on DD. The simulation domain is partitioned into sub-domains of equal size and dimension. Each sub-domain calculation is distributed to a single processor. All the processors associated with individual sub-domains perform the calculation the same way as the equivalent serial simulation, but with smaller domain sizes. The complexity arises from the interface cells between sub-domains, as there can be flow exchange between adjacent cells across the sub-domain interface at each time step. Similar to the principle underlying MPI, the communication between adjacent interface cells is handled through message passing. Adjacent sub-domains communicate with each other through a central server in order to maintain synchronization. The first variable that needs to be passed through the sub-domain interfaces is the volume of water (Q) transferred from one domain to another at each pair of adjacent cells through the interface. This, along with the sub-domain internal flow routing calculated at each computational node, is used to update the water depths of the interface cells.

The volume of flow exchange (*Q*) is not the only variable that needs to be passed between sub-domain interfaces at each time step. In order to calculate how much water will be passed through the interfaces during the next time step, each sub-domain needs to acquire the energy slopes along the interface cells to the adjacent sub-domains for the current iteration. Therefore, energy slope is another variable that is updated and passed through adjacent domains. At each time step, each computational node sends the interface variables to the remote server. Once all the data have been received by the server, it calculates the slopes across the sub-domain boundaries. After this calculation, flux from and slopes to the adjacent sub-domains can be sent to each individual sub-domains. The server enters the waiting state and this initiates the next iteration for each node.

Another important consideration in diffusion-based model is the time step used. FloodMap allows two types of time step to be specified: (i) an explicitly fixed time step that ensures model stability throughout the simulation; and (ii) an implicit time step calculated during run time based on either Courant–Freidrichs–Levy condition or adaptive time-stepping (Hunter *et al.*, 2005). As the focus of this paper is the comparison of computational performance between simulations, in order to eliminate the inherent effects of faster execution time associated with a coarser mesh should an implicit time step be used, a fixed time step is used for all the simulations. This is specified as 1 s based on previous simulations and computational stability is maintained in all the simulations carried out.

3. Case studies

The parallelized model was tested by comparison with previous simulations carried out for three river reaches in the UK: the River Wharfe in Yorkshire (Tayefi *et al.*, 2007); the River Ouse near where it intersects the road A64 (Yu and Lane 2006a); and the River Ouse where it passes through the City of York (Yu, 2005). The first two reaches are located in relatively rural areas. The third reach, i.e. the river reach across the city centre of York is an urban reach with extensive linear and blockage features adjacent to the river channel. The reaches and the associated floods modelled are summarized in Table 1. This allows the behaviour of the parallelized model to be evaluated in environments with a range of topographic and flow conditions. Moreover, it is recognized that the performance of the model may exhibit a functional relation with the number of sub-domains, the way a domain is partitioned and the number of cells. Therefore, for each reach, meshes of various resolutions and DD configuration patterns were constructed to take into account these factors. All the floods simulated occurred in the year 2000. Different flow boundary conditions were used. Reach 1 used flow predictions obtained from a 1D hydraulic model (Hec-Ras, Tayefi *et al.*, 2007). For Reach 2, given its relatively small size, a uniform flow has been assumed in the river channel (c.f. Yu and Lane 2006a). Simulations at Reach 3 have been generated by a version of the model tightly coupled with a full 1D solution of the Saint-Venant equations in the river channel (Yu, 2005).

4. Results

The parallel solver has been tested on a 160-processor 64-bit Itanium cluster at Loughborough University. The cluster consists of 20 computational nodes, each having four dual-core Itanium 1.6GHz CPUs and 16GB of main memory. The computational nodes are interconnected by a high-performance Quadrics communication network. The sustained bandwidth of the network is >900MB/s and the latency is ~1.5μs. In total 1019 simulations were run, distributed among various numbers of nodes (up to 64). These are summarized in Table 2, together with the total run time for each simulation. As the cluster has a 100-hour limit on the wall time, some of the simulations did not finish. In such cases, the estimated time to finish the run is presented. Only some of the simulations in Reach 3 (City of York) require the estimation of total run time based on model performance obtained from the first 100 hours. For this particular reach, the estimation of total run time based on the model performance during the first 100 hrs is considered to be a valid approach as:

(i) a fixed time step is used for all the simulations; (ii) the maximum computational requirement has been reached during the first 100 hours so the model performance is unlikely to vary considerably after that; and (iii) the focus of this study is on the relative performance of the simulations. These simulations enable the parallelized model to be evaluated in terms of both the accuracy and the computational performance. This is discussed in Section 4.1 and 4.2 respectively.

4.1 Model accuracy test

To investigate the general performance of the parallelized model in terms of its predictive accuracy of key flow variables, water depth and inundation extent obtained from the parallelized simulations are compared with those obtained from the serialized simulations. This was carried out for simulations with different number of sub-domains in each reach. Results show that, for all the parallelized simulations, as expected, there is no difference in the predicted water depth and wetting front when compared with the results obtained from the original serialized model. Mass balance is maintained in all cases. Results obtained for Reach 1 are presented in Figure 1 to illustrate this. It shows predicted water depths within Reach 1 obtained from the serialized model (Figure 1a) and those generated by the parallelized model with 1_2 (Figure 1b), 1_4 (Figure 1c), 2_4 (Figure 1d) and 1_8 (Figure 1e) DD configuration patterns. Sub-domain boundaries are indicated with a fishnet and water depth predictions in each sub-domain are distinguished using different colour schemes. Close examination of the interface cells in the parallelized simulations reveals that connectivity is maintained well at the sub-domain boundaries (e.g. between sub-domain 3 and 7 in Figure 1c). This is characterised by smooth water surface slopes between adjacent cells across sub-domain interfaces, corresponding with those observed in the serialized case, indicating that the flow is being transferred through sub-domain common boundaries in an appropriate way.

4.2 Performance test

Various mesh resolutions and DD configuration patterns were used to test the computational performance of the parallelized model. Model performance is investigated using two common measurement parameters, i.e. speedup and efficiency, in relation to the number of sub-domains, the DD configuration pattern and the total number of cells.

Parallel performance – speedup and efficiency

A common parameter to the measurement of computational efficiency of parallelism is the speedup of the model which is defined as the ratio between the uni-processor execution time to that of the multi-processor execution (Equation 7).

$$S(N,P) = \frac{T_{seq}(N)}{T(N,P)} \qquad (7)$$

where $T(N, P)$ is the runtime of the parallel algorithm, and $T_{seq}(N)$ is the runtime of the sequential algorithm. In relation to this, another performance metric is the parallel efficiency $E(N, P)$ which, for a problem of size N on P nodes is defined by:

$$E(N,P) = \frac{S(N,P)}{P} \qquad (8)$$

For the specific modelling of flood inundation, the actual model performance over time for a single simulation will vary with the number of active wet cells that needs to be calculated. It is anticipated that, at the beginning of the simulation when no floodplain active wet cells exist, the performance would be similar for the parallelized and serialized models, as no calculation of floodplain flow is required in both cases. Indeed, in all the flood events simulated, the floodplain is inundated almost from the very beginning of the hydrograph. Therefore the period of time the floodplain is not inundated accounts for very little of the overall runtime. The performance of the parallelized model may not exhibit much computational advantage over the serialized model until the number of active wet cells is sufficiently high. This indicates the limitation of using a single total run-time to evaluate the performance of this type of model. Therefore, rather than using a single total run-time to evaluate the model performance, this is investigated here with a parameter termed as “performance ratio” over time, defined as the ratio between the actual computational time used to simulate a specified period of flood event to the length of this period. For all the simulations carried out, this parameter is calculated every 60 seconds, thus allowing the performance ratio to be presented as a time series for each sub-domain simulation. Figure 2 shows the performance ratio over time for simulations carried out with 1, 2, 4 and 8 processors in Reach 1 with a 4 m mesh. As expected, performance degrades for all simulations as the number of wet cells increases, but the rate of degradation becomes progressively smaller as the number of processors increases (Figure 2a). Model speedups are calculated based on the performance ratio defined above at the fixed 60-second interval. The results show that, for most of the time, the speedups improve with the increased number of processors. The maximum speedups achieved for the 2-node, 4-node and 8-node simulations are 1.75, 1.98 and 2.71 respectively, when translated into efficiency, corresponding to 0.87, 0.50 and 0.33.

The overall speedup and efficiency for all the simulations were calculated from Table 2. These are shown in Figure 3 with the average value plotted for simulations with the same number of nodes but different DD configuration patterns. At the small end of the number of distributed processors (2, 4, and 8 nodes), the speedups and efficiencies exhibit a broadly similar but slightly lower magnitude to those obtained in Neal *et al.* (2009). According to Amdahl’s law, the maximum speedups that can be achieved given the fraction of serial code that can be parallelized, here denoted as *P*, and the number of distributed nodes *N*, can be calculated as:

$$S = \frac{1}{(1-P)+(P/N)} \tag{9}$$

Profiling of the parallelized simulations suggests the fractions that were actually parallelized varied between 92.7% and 95.6% for the simulations carried out. This gives the maximum speedups for the number of nodes utilized in the simulations according to Amdahl's law (Table 3). Table 3 also listed the best speedups achieved in this study for the corresponding simulations in Table 2. Comparing these with the speedups achieved in this study, strong variations are noted, with speedups approaching or exceeding the theoretical values in some cases, while, in most cases, the speedups are below the theoretical values.

A number of tentative observations can be made here which will be examined in more detail in the subsequent sections. First, it is generally the case that given the same number of cells, simulations with more nodes achieve higher speedups, and the spread of speedups seems to be positively associated with the number of cells. Second, given the same number of nodes, the speedups appear to be a function of the number of cells in the simulations. Third, high speedups are generally associated with low efficiencies. For example, the 2-node simulations are the most efficient. The maximum speedup of nearly 16 was achieved with the 64-node simulation of Reach 3 with a 4 m mesh and an efficiency of around 0.24.

Thread profiling

The model structure of FloodMap is similar to that of LISFLOOD-FP which is reported in Neal *et al.* (2009), so it will not be covered here. Figure 4 shows the average percentage of computational time utilized by the major components of the serial model for a simulation carried out for Reach 1. The general pattern is in line with what was described by Neal *et al.* (2009). The major components of the serialized model are the floodplain flow calculation, drying check, boundary condition setup and time step control, with floodplain flow calculation accounting for between 36% and 58% of the total computational time during most of the simulation. The second time consuming element is the drying check, which essentially puts a limit on the amount of outflow from each cell, so that negative flow depth does not occur. It is computational expensive because a change in outflow for one cell means a change in inflow for the other. Thus, this has to be calculated iteratively across the domain until the criteria is met for all the active cells in order to maintain mass conservation. This can take up to 19.5% of the total run time of this simulation. Apart from this, up to 16% of the time is spent on setting up the flow boundary conditions, due to the nature of the inflow type for this particular simulation (multiple hydrographs). Time step control takes up to 8.4% of the computational time.

A major processing component is introduced to the model when the code is parallelized through domain decomposition. This is associated with the communication between sub-domains at each time step through the remote server. Communication between sub-domains in order to maintain synchronization is regarded as one of the key penalties associated with distributing an application over an array of independent processors (Keyes, 1989). The amount of time involved in the node-server communication is measured by a parameter termed "communication overhead", defined as the time taken to receive the updated server data after it sends out its boundary data to the server divided by the total time used to run an iteration. Figure 5a shows the computational profiles over time for the major components of the parallelized version of the simulation in Figure 4, with a 2_4 DD configuration pattern. Figure 5b compares the average of each major computational component over the whole simulation for both the serialized (Figure 4) and the parallelized (Figure 5a) simulations. The results suggest that the communication overhead is a major computational component for this particular simulation, accounting for around 50% of the total run time. It also shows that parallelization does not change the relative weighting of the existing components. In addition, the relative weighting of the computational time for communication increases over time, likely due to the increase in the load imbalance between processors. The fluctuation of the curves compared to Figure 4 is probably due to the synchronization of the sub-domains computations.

Load balancing and communication overhead

One of the key considerations in parallel modelling using domain decomposition is the load balance between parallelized sections. For the time-evolving type of modelling like flood inundation, the requirement of synchronization between processors is essential in order to achieve mass conservation. This has a bearing on the model efficiency, as the imbalance of calculation time between processors means the model will have to wait for the slowest sub-domain calculation to finish in order to proceed to the next time step. The performance of a parallel model can be significantly degraded if the computational time for each sub-domain is heavily unbalanced. This may affect both the general performance of the model and the computational profiles of the model simulations. This involves the communication between nodes and server. For ease of analysis here, the communication overhead defined in the last section is further broken down into three components. The first component is the server side waiting time, i.e. the interval between the time the server receives the first and last updated set of data from the nodes. The second component is the time the server takes to process the interface data before it sends the updated data to all nodes. The third component is related to the time directly involved in the message passing between the node and server. This is determined by the network speed and the amount of data passed through. Whilst the network speed may be a result of a number of factors and may not be known a *priori*, the later is essentially related to the number of inter-domain interface cells, which, in turn, is a function of how a domain is partitioned. The calculation time in the server and the time taken for message passing between the nodes and the server through the network are expected to be minor compared with node-side calculation and the associated server-side idle time. No attempt is made here to separate these three components. Rather, this study has used the node-side waiting time as a surrogate for the aggregation of the imbalance between the sub-domain calculations and the corresponding network issues.

In diffusion-based flood inundation models, sub-domain calculation time is associated with the number of cells that needs to be calculated for each sub-domain i.e. the number of wet cells on the floodplain at each single time step. The approach to domain decomposition utilized in this study will always produce sub-domains of equal dimension, thus implicitly the same number of maximum computational cells for each sub-domain, if the whole floodplain is inundated, theoretically resulting in a perfectly balanced load. Given that the amount of calculation required at each sub-domain is likely to be different at different stages of the simulation, the load imbalance will exist across sub-domains. This is expected to be minimized at the start of the simulation when there are no active floodplain flow-routing cells. As the floodplain is gradually wetted, the number of cells that needs to be dealt with will increase, potentially disproportionately for different sub-domains. To investigate this, the average waiting time taken for a sub-domain to receive updated data from the server is calculated at intervals of 60-second. Figure 6 plots the waiting time of each sub-domain for a simulation at Reach 1 with a 2_4 DD configuration pattern and a 4 m mesh. A marked difference between the waiting time across sub-domains can be observed. As expected, this is negatively correlated with the number of wet cells predicted at the end of the simulation, which is used here as a crude indicator of the computational requirements for each sub-domain. Notably, sub-domains 1, 2 and 7, which have the greatest number of cells to compute take the least amount of time to wait, indicating a faster execution and, therefore, longer waiting time for sub-domains with less wet cells to compute.

Therefore, the load balancing can be related to how the domain is partitioned. Two parameters can be used to describe how the domain is partitioned: domain granularity and DD configuration pattern. The total number of sub-domains partitioned for a domain is commonly termed "domain granularity". In order to get the same number of sub-domains, there can be a number of ways to partition a domain. This is here termed "DD configuration pattern". Two considerations arise and need to be investigated in relation to these two parameters. First, as the degree of domain granularity increases, the overall communication overhead will increase, resulting in less efficient parallelization. Davis and Peter Sheng (2003) found, in their modelling application using DD and MPI, that there is a threshold value beyond which the parallelized model performance will actually begin to decline due to the increase of the relative weighting of the communication overhead. The effect of domain granularity on model performance is a function of the total number of cells used in the model. For simulations with the same domain granularity, the weighting of the communication overhead will decrease with a larger domain. Second, how the decomposition is configured may have an impact on the performance of the parallelized model. There may be a number of ways to partition a simulation domain into the same number of sub-domains. For example, Figure 7 shows the possible ways of partitioning a simulation domain into 6 equal-sized rectangular sub-domains. These different DD configuration patterns might be associated with different amounts of data to be processed and communicated. First, this has an implication in terms of the amount of data to be computed in each sub-domain. This is illustrated in Figure 7 where sub-domains with equal size and dimension can have big differences in the amount of cells to be computed. Second, the amount of data to be passed between node and server can be different. Suppose that the domain is a square. If the number of cells along each side of the domain is denoted as L, the total number of cells that needs to be passed to and from the server for the sub-domains at each time step is $3L$ for Figure 7a and Figure 7d, while for the Figure 7b and Figure 7c, this number is $5L$. This may have an impact on the parallelized model, as has been shown above, node-side imbalance can reduce the model performance. A number of simulations were carried out to investigate the efficiency of parallelism in relation to domain granularity, DD configuration and number of cells.

Effect of domain granularity

To investigate the effect of granularity on model performance in relation to the number of cells, simulations at each reach with different domain granularity and mesh resolutions were designed. For Reach 2, this was carried out with a 2 m, 4 m and 8 m mesh, corresponding to ~800k, ~200k and ~50k cells respectively. Figure 8a plots the overall speedups for the parallelized simulations carried out with a 2 m, 4 m and 8 m mesh, distributed over 2, 4, 8, 16 and 32 processors respectively. Two observations arise. First, the general pattern for the 4 m and 8 m simulations is that the model achieves higher speedups with the increase in the number of processors used, but only up to a threshold value, beyond which, there is a decline in the speedups. For example, the 4 m simulations achieve the highest speedup with 16 nodes. But with 32 nodes, the speedups decline to a level that approximates that of 8 nodes. Second, speedups for simulations with the same number of nodes tend to be higher for finer meshes (i.e. the 2 m mesh). This is in line with what Neal *et al.* (2009) found for this type of model with the shared-memory OpenMP approach. This can be explained by the percentage of communication overhead plotted in Figure 8b. It shows that, as the mesh is gradually coarsened, the percentage of time spent on message passing is increased proportionately. Similarly, as the number of sub-domains increases, the relative weighting of communication time is also increased. This occurs as a function of the number of cells that needs to be passed through interfaces in relation to the total number of cells that needs to be computed in each sub-domain. Simulations carried out for Reaches 1 and 3 reveal a similar pattern.

Effect of DD configuration pattern

To investigate the effect of DD configuration pattern, the overall speedups are calculated for simulations of Reach 2 with 8 sub-domains comprising different configurations and with a 2 m, 4 m and 8 m mesh respectively. Figure 9 plots the speedups for all the simulations. Apart from the observation that a finer mesh produces larger speedups, the results show that there can be some differences between the speedups achieved by simulations with the same mesh resolution and different DD patterns.

The mean percentage communication overhead for each DD configuration shown in Figure 9 is plotted in Figure 10 together with the corresponding standard deviations. It shows that there can be big differences in the communication overhead, both for simulations with different DD configuration patterns and across the sub-domains for the same simulation. In particular, for this specific application, the 8_1 configuration has the largest variation while the 4_2 configuration has the smallest. Similar results are also found for other reaches and with different number of sub-domains. It may be expected that simulations with larger variation in the communication overhead weighting will result in smaller speedups. However, when Figure 10 is compared with Figure 9, such association is not found. This may need to be interpreted in the context of the domain size in relation to the DD configuration.

5. Conclusion and future developments

In this paper, a 2D diffusion-based flood inundation model (FloodMap) has been parallelized using the domain decomposition approach based on MPI, where each computational node performs the same calculations and variables are passed through interfaces of adjacent sub-domains. The model maintains mass conservation and produces the same results as the serialized model. Speedups and efficiencies have been used to measure the computational performance of the model in relation to the number of cells, the pattern of DD configuration and the domain granularity.

Results show that this approach is more effective with simulations of a relatively large number of computational cells (>100k) and this can be regarded as a favourable feature of a parallel model intended for simulations with ever-increasing domain size and resolution. In such cases, there is a positive relation between the maximum speedups achieved and the number of the processors used. But when the number of cells decreases, there will be degradation in the slope of the speedups. This is shown to be associated with the communication overhead incurred in the parallelized model, which can account for a significant amount of the model run time. Although the overall speedups for some simulations are not ideal according to Amdahl’s law, this development is promising as it allows flood inundation modelling in situations where large spatial scales are involved or high-resolution topographic data are necessary.

It was found in this study that the communication overhead and load imbalance are two major bottlenecks in applying this approach with higher domain granularity. Therefore, further development of this approach will focus on these two essentially interrelated aspects: (i) minimizing the communication overhead between nodes; and (ii) more effective domain decomposition approach that takes into account the balance of computational loads between nodes. As the current parallelization does not rely on specific MPI software, there is scope for these two limiting factors to be addressed through code modification in order to realize the full potential of parallelization. In the current parallelization implementation, message exchange between nodes is achieved through a central server. To minimize the communication overhead, instead of a three-way communication through the server, a direct two-way communication between nodes might be an alternative. Depending on the structure of the HPC clusters, the implementation of such an approach will need to know how a cluster distributes the nodes among its resources. This can be further related to the second limiting factor, i.e. the load imbalance. Communication overhead essentially arises from the imbalance between computational loads among distributed nodes. Requirement of synchronization implies a faster node (e.g. nodes 4, 6 and 8 in Figure 1d) will need to wait till the slowest node (e.g. node 1 in Figure 1d) to finish before proceeding to the next time step. Two approaches might be used to address this problem. First, a non-uniform DD pattern (interactive/dynamic DD) can be used to minimize the potential load imbalance between nodes at the initial domain partitioning stage. This can be carried out, for example, through interactive domain partition by a user with prior knowledge about the likely places to be inundated. Again, using Figure 1d as an example, based on the inundation pattern observed towards the end of the simulation, domains 1, 2 and 7 might need to be further partitioned such that the potential computational loads are best balanced between nodes. However, this also needs to consider the dynamic nature of the flow: the load imbalance can occur both spatially and temporally. Therefore, the seemingly best domain partitioning may not be the best one throughout the model run. From this perspective, an adaptive DD that adjusts the amount of elements in each sub-domain during run time might offer better potential. One way of implementing such a scheme is to keep the current message exchange structure and add further functions to the central server to allow adaptive DD pattern and data re-distribution during run time. The development in these two aspects is expected to improve the model computational performance significantly, which in turn will allow the mismatch between the abundance of high-resolution topographical data and the usual practice of data coarsening in 2D flood inundation models to be addressed.

One last observation from this study is that as currently there is no industry standard Java MPI library available, the development time of parallelization through DD based on message passing can be much longer compared to approaches such as those based on OpenMP. However, once developed, it may offer better speedups and efficiencies (e.g. Neal, *et al.* 2010). This is especially valuable for flood inundation modelling in urban applications where high-resolution topographical data is required, in particular over large spatial and temporal scales and in situations where large numbers of simulations are required, for example, when evaluating uncertainty under climate change scenarios.

Acknowledgements

The author wishes to thank Jeffery Neal and Tim Fewtrell at Bristol University and two anonymous reviewers who provided very useful comments on the paper. The author also thanks the High Performance Computing Centre at Loughborough University for the technical support.

References

Bates, P.D. and De Roo, A.P.J., 2000. A simple raster-based model for flood inundation simulation. *Hydrological Processes* 236:54-77.

Bates, P.D., Marks, K.J. and Horritt, M.S., 2003. Optimal use of high-resolution topographic data in flood inundation models. *Hydrological Processes* 17:537-557.

Bradbrook, K.F., Lane, S.N., Waller, S.G. and Bates, P.D., 2004. Two dimensional diffusion wave modelling of flood inundation using a simplified channel representation. *International Journal of River Basin Management* 2(3):1-13.

Davis, J.R. and Peter Sheng Y., 2003. Development of a parallel storm surge model. *International Journal for Numerical Methods in Fluids* 42:549-580.

Fewtrell, T.J., Bates, P.D., Horritt, M.S. and Hunter, M.N., 2008. Evaluating the effect of scale in flood inundation modelling in urban environments, *Hydrological Processes*, 22: 5107-5118.

Hervouet, J., 2000. A high resolution 2-D dam-break using parallelization. *Hydrological Processes* 14(13): 2211-2230.

Hunter, N.M., Horritt, M.S., Bates, P.D., Wilson, M.D., Werner, M.G.F., 2005. An adaptive time step solution for raster-based storage cell modelling of floodplain inundation. *Advances in Water Resources* 28 (9): 975–991.

Keyes, D.E.,1989. Domain decomposition methods for the parallel computation of reacting flows. *Computer Physics Communications* 53:181-200. Redwood City, CA.

Lane, S.N., Reid, S.C., Tayefi, V., Yu, D. and Hardy, R.J., 2008. Reconceptualising coarse sediment delivery problems in rivers as catchment-scale and diffuse, *Geomorphology*, 98(3-4):227-249.

Lane, S.N., Tayefi, V., Reid, S.C., Yu, D. and Hardy, R.J., 2007. Interactions between sediment delivery, channel change, climate change and flood risk in a temperate upland environment, *Earth Surface Processes and Landforms* 32(3):429-446.

Marks, S. and Bates, P.D., 2000. Integration of high-resolution topographic data with floodplain flow models. *Hydrological Processes* 14:2109-2122.

Neal J.C., Fewtrell, T.J. and Trigg, M., 2009. Parallelization of storage cell flood models using OpenMP. *Environmental Modelling & Software* 24:872-877.

Neal J.C., Fewtrell, T.J. Bates, P.D. and Wright N.G. 2010. A comparison of three parallelization methods for 2D flood inundation models. *Environmental Modelling & Software* 25:398-411.

Rao, P., 2005. A parallel RMA2 model for simulating large-scale free surface flows. *Environmental Modelling & Software* 20(1):47-53.

Tayefi, V., Lane, S.N., Hardy, R.J. and Yu, D., 2007. A comparison of 1D and 2D approaches to modelling flood inundation over complex upland floodplains, *Hydrological Processes* 21:3190-3202.

Yu, D., 2005. Diffusion-based flood modelling over topographically complex floodplains. *PhD Thesis*. Department of Geography, University of Leeds.

Yu, D. and Lane, S.N., 2006a. Urban fluvial flood modelling using a two-dimensional diffusion wave treatment, part 1: Mesh resolution effects, *Hydrological Processes* 207:1541-1565.

Yu, D. and Lane, S.N., 2006b. Urban fluvial flood modelling using a two-dimensional diffusion wave treatment, part 2: Development of a sub grid-scale treatment. *Hydrological Processes* 207: 1567-1583.

List of Tables

Reach	Reach description	Flood duration	Flow boundary condition
Reach 1: River Wharfe	Rural reach	10 hrs	Flow obtained from 1D model HecRAS.
Reach 2: River Ouse A64	Rural reach	50 hrs	Uniform flow.
Reach 3: River Ouse City of York	Urban reach	300 hrs	Tightly coupled with 1D solution of Saint-Venant equations.

Table 1: Study reaches and the associated flood characteristics.

			Parallelized Model Run Time														
	Resolution	No. of cells	1 node	2 nodes		4 nodes	8 nodes				16 nodes					32 nodes	64 nodes
			1_1	1_2	2_1	2_2	1_8	8_1	2_4	4_2	4_4	2_8	8_2	1_16	16_1	4_8	8_8
Reach 1 River Wharfe	4 m (10hrs)	~232k	10.0	6.9	6.9	5.45	3.83	4.14	3.64	4.2	3.88	3.52	3.83	3.42	3.98	3.55	
	8 m (10hrs)	~58k	3.13	2.59	3.55	2.37	2.12	2.72	2.6	2.5	2.7	2.70	2.83	2.65	2.99	3.03	
Reach 2 River Ouse	2 m (50hrs)	~800k	91.7	72.2	75.5	45.4	30.0	28.0	31.2	27.0	22.0	23.0	23.0	22.0	20.8	18.6	
	4 m (50hrs)	~200k	36.9	22.5	21.0	15.8	14.0	13.0	13.0	13.0	12.0	11.0	12.0	11.7	11.1	12.1	
	8 m (50hrs)	~50k	12.3	9.32	9.8	8.46	7.5	6.8	8.2	7.5	7.5	8.3	8.1	8.1	7.9	9.57	
York	4m (300hrs)	~1000k	621(E)	431(E)		281(E)			194 (E)			157(E)				64.5	39.7
	8m (300hrs)	~250k	166(E)	154(E)		106(E)			68.7			54.2				41.2	

Table 2: Simulations carried out and the run time taken for each simulation with different mesh size and DD configuration pattern. E: unfinished simulation with estimated run time. Shaded cells are non-run simulations. Unit is in hour.

No. of nodes / % parallelized	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
92.7%	1.86	3.28	5.29	7.64	9.81	11.43
95.6%	1.92	3.53	6.12	9.64	13.54	16.97
Best achieved	1.64	2.34	3.39	4.41	9.62	15.64

Table 3: theoretical speedups calculated according to Amdahl's law and the best speeds achieved.

List of Figures

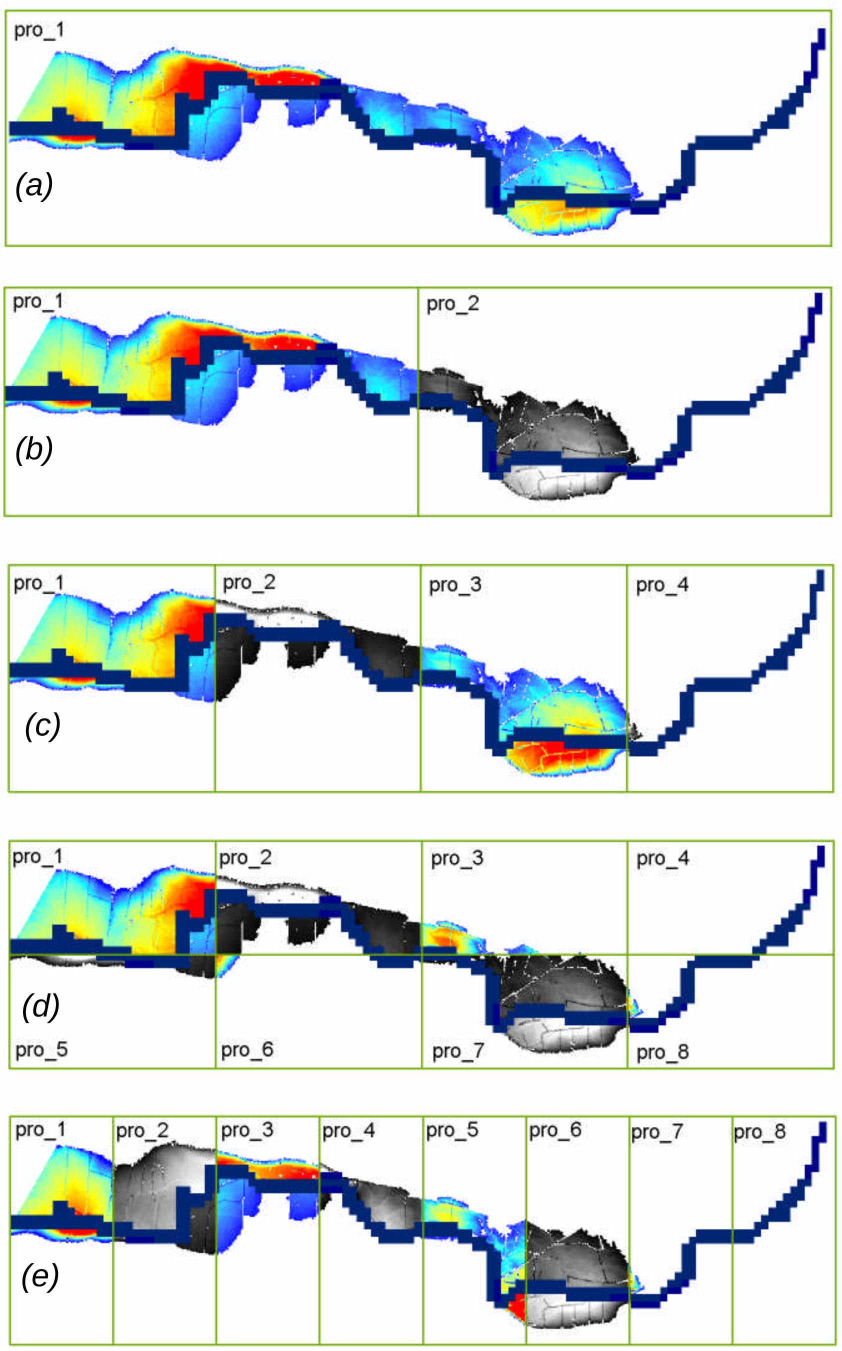
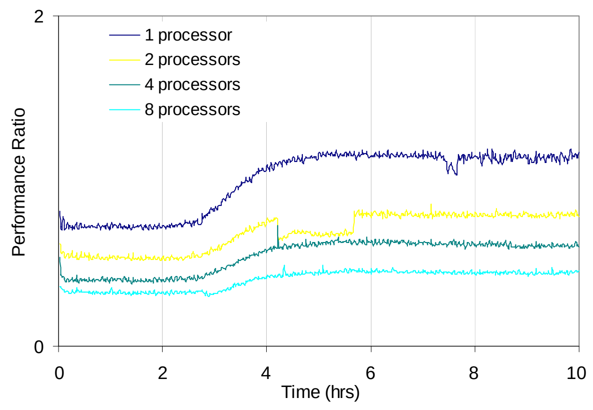
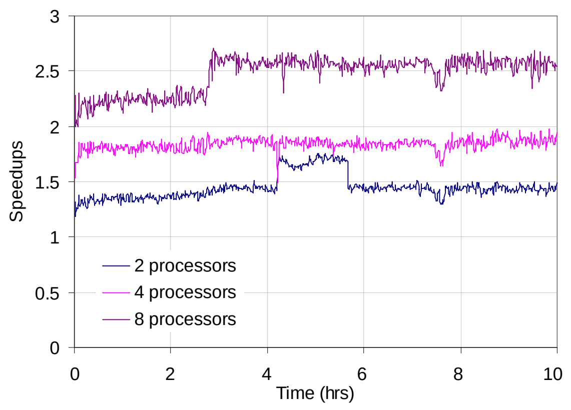


Figure 1: Predicted water depths at 10 hrs at Reach 1, obtained from: (a) the serialized simulation; (b) the parallelized simulation with a 1_2 DD ratio; (b) the parallelized simulation with a 1_4 DD ratio; (c) the parallelized simulation with a 2_4 DD ratio; (d) the parallelized simulation with a 1_8 DD ratio.

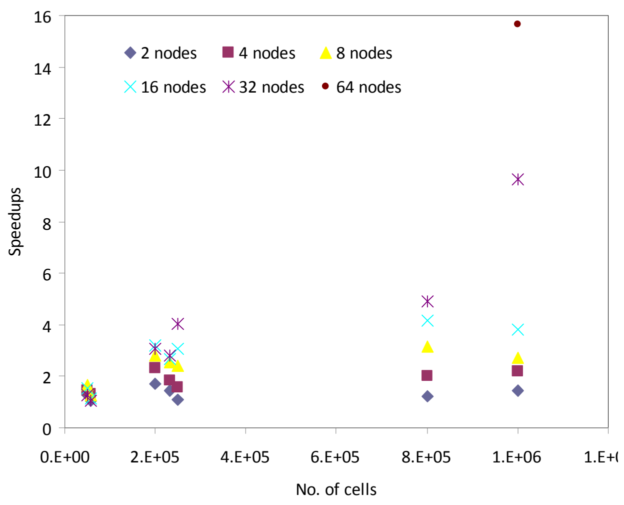


(a)

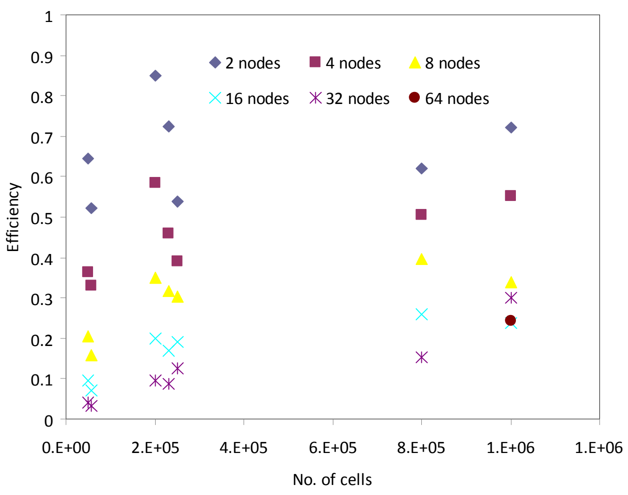


(b)

Figure 2: (a): performance ratio over time for simulations with 1, 2, 4 and 8 processors. (b): speedups over time compared with the serialized simulation for the parallelized simulations in (a). The simulations were carried out at Reach 1 with a 4 m mesh.



(a)



(b)

Figure 3: Speedups (a) and efficiency (b) achieved with the simulations listed in Table 1. Where there are different DD configuration patterns, the averaged values are plotted.

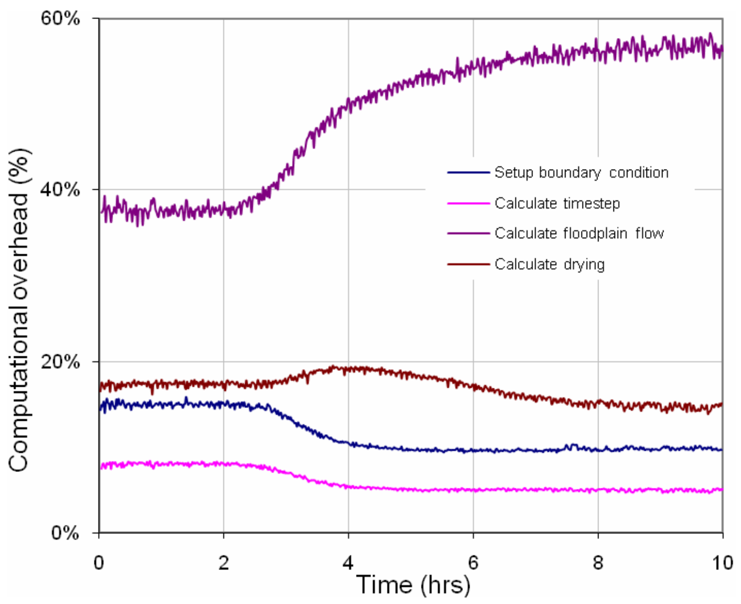
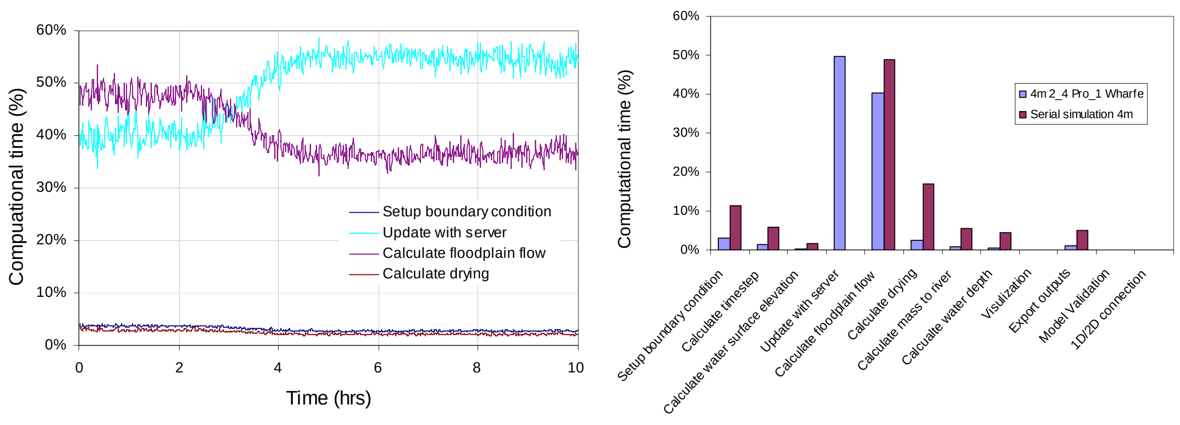


Figure 4: Major computational components of the serialized model. The simulation was carried out at Reach 1 with a 4 m mesh.



(a) *(b)*

Figure 5: (a) Major computational components of the parallelized model. The simulation was carried out at Reach 1 with a 4 m mesh and a 2_4 DD configuration; (b) comparison of the major computational components between the serialized and parallelized simulations.

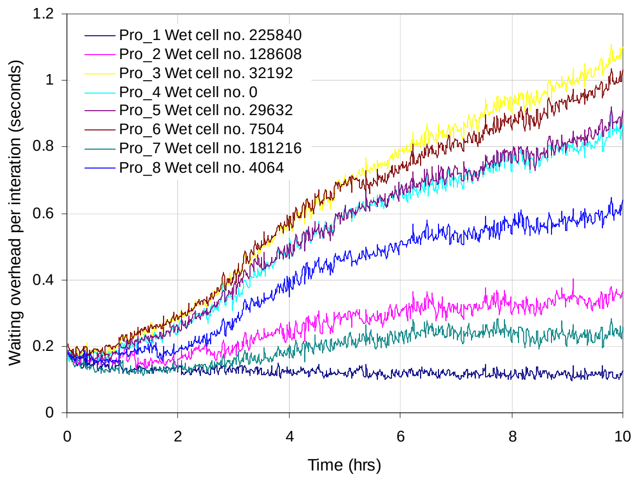


Figure 6: Node-side waiting time for different sub-domains in the simulation with a 2_4 DD ratio at Reach 1 (Figure 1c).

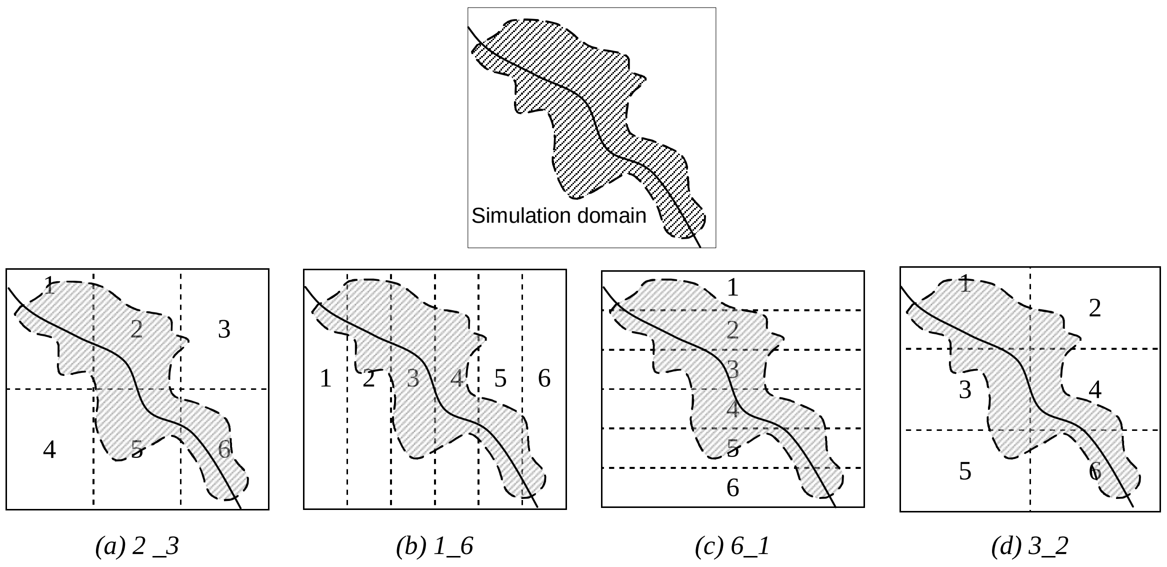
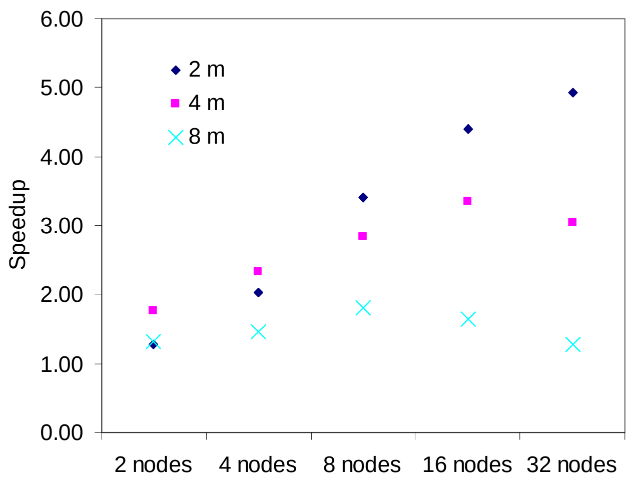


Figure 7: Ways of spatial decomposition of a simulation domain into six sub-domains where the number of sub-domains is equal to the number of distributed processors targeted. Short dotted lines are the sub-domain interfaces. The DD configuration pattern is indicated with row_column as labels. Shaded areas are the potential maximum inundation extents. Solid line across the domain is the location of the river channel.



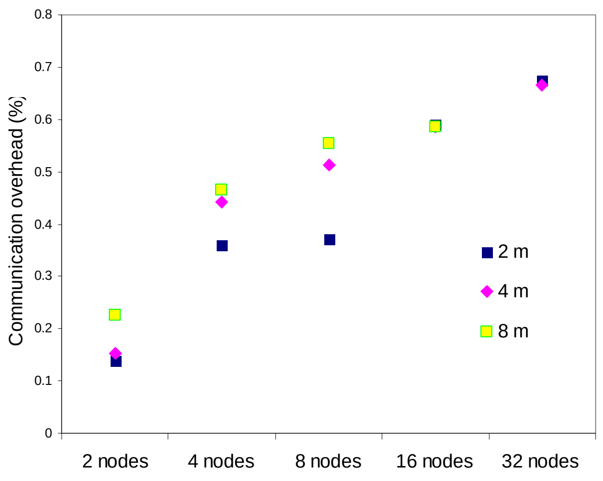


Figure 8: Effects of granularity on model performance in relation to the mesh resolution and the number of sub-domains. The speedups and communication overhead are averaged for different DD configuration patterns used in Table 2.

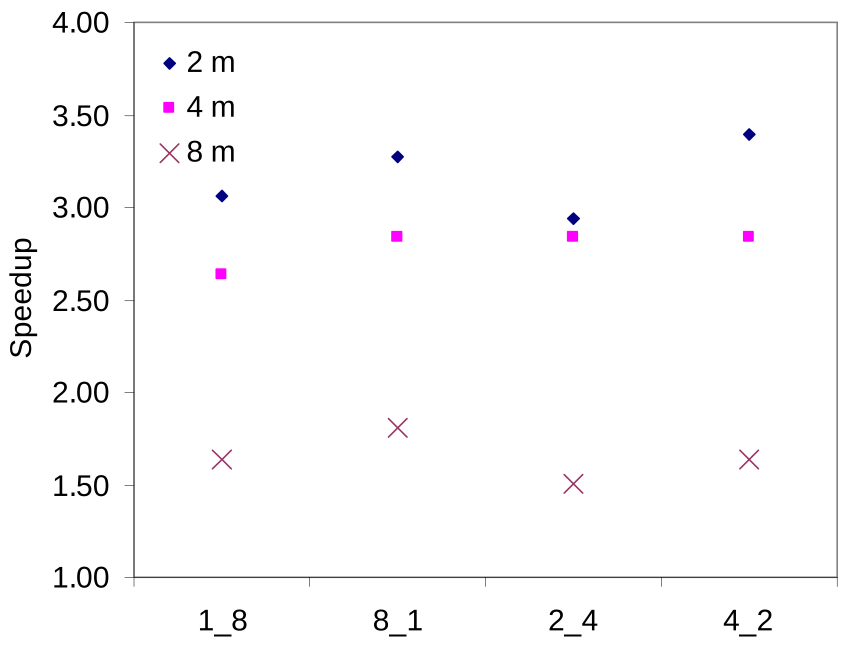
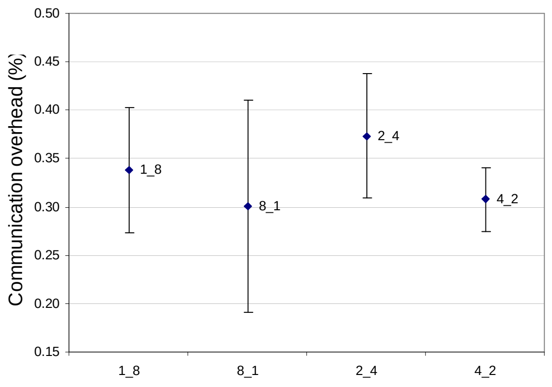
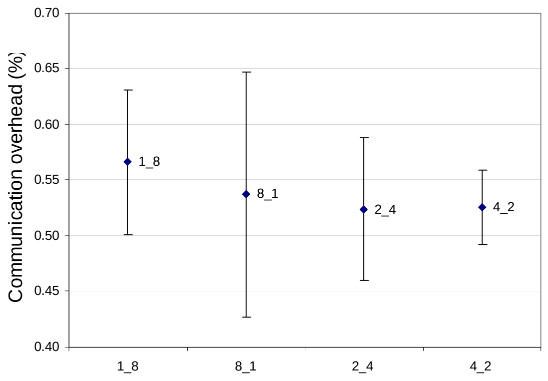


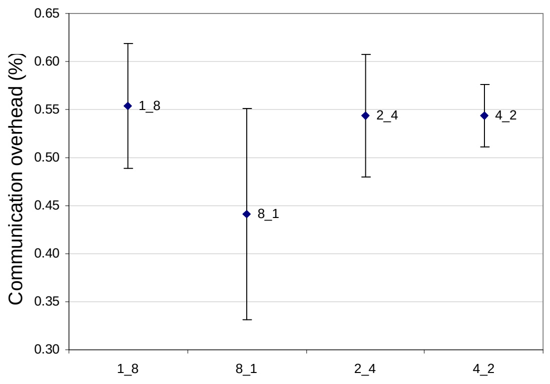
Figure 9: Speedups for simulations with 8 sub-domains at Reach 2 with different DD configuration patterns.



(a) 2 m



(b)4 m



(c) 8 m

Figure 10: Communication overhead for parallelized simulations with different DD configuration patterns: (a) 2 m, (b) 4 m and (c) 8m at Reach 2. Bars encompass 1 standard deviation on each side of the point.