A Hybrid Parallelization Approach for High Resolution Operational Flood Forecasting

Swati Singhal, IBM Research India, New Delhi, India, swatisin@in.ibm.com

Lucas Villa Real, IBM Research Brazil, Sao Paulo, Brazil, lucasvr@br.ibm.com

Thomas George, IBM Research India, Bangalore, India, thomasgeorge@in.ibm.com

Sandhya Aneja, Universiti Brunei Darussalam, Brunei Darussalam, sandhya.aneja@ubd.edu.bn

Yogish Sabharwal, IBM Research India, New Delhi, India, ysabharwal@in.ibm.com

***Abstract*—Accurate and timely flood forecasts are becoming highly essential due to the increased incidence of flood related disasters over the last few years. Such forecasts require a high resolution integrated flood modeling approach. In this paper, we present an integrated flood forecasting system with an automated workflow over the weather modeling, surface runoff estimation and water routing components. We primarily focus on the water routing process which is the most compute intensive phase and present two parallelization strategies to scale it up to large grid sizes. Specifically, we employ nature-inspired decomposition of a simulation domain into watershed basins and propose a master slave model of parallelization for distributed processing of the basins. We also propose an intra-basin shared memory parallelization approach using OpenMP. Empirical evaluation of the proposed parallelization strategies indicates a potential for high speedups for certain types of scenarios (e.g., speedup of 13× with 16 threads using OpenMP parallelization for the large Rio de Janeiro basin).**

I. Introduction

Operational flood forecasting is becoming increasingly important due to the changing global climate and frequent incidence of flooding-related catastrophes in recent years [1]. Flooding incidents affect public safety, damage infrastructure, disrupt transportation networks, contaminate drinking water, and interfere with the smooth operation of first responders to more serious emergencies resulting in massive socio-economic losses. Studies [2] indicate that accurate and timely predictions accompanied by judicious response can go a long way in mitigating the impact of flood related disasters.

Depending on the geography and terrain, there are multiple causes for flooding, of which the most common are: (a) sudden heavy precipitation incidents in urban areas with poor drainage, (b) seasonal storms resulting in heavy persistent rainfall that saturates land and results in overflowing waterbodies, (c) tidal surges and breached levees due to wind storms in coastal areas. Since each of the above causes is intimately connected with unpredictable short-to medium range weather events, flood modeling capabilities in the past were limited by the availability of accurate weather observations. In recent years, there has been a heavy focus on integrated forecasting of weather (precipitation, temperature, wind speed) and flooding events. Typically, these integrated forecasting systems follow a two stage mechanism where the first stage employs a high resolution localized weather model to predict precipitation with sufficient lead time. The second stage uses these predictions as input to an overland flood routing model, which computes surface runoff and routes the flow taking into account various surface characteristics such as variation in land use type, topography, vegetative cover, etc. In such systems, the weather forecasting is performed using fine resolution regional and global atmospheric models that discretize the nonlinear partial differential equations representing evolution of atmospheric flows in time [3], while the overland flows are simulated via equations based on conservation of mass and momentum with the vertical effects simplified to yield two-dimensional shallow water equation [4]. The overland flood routing engine could also be used to provide input flow at various pour points along a river to enable a dynamic river model determine the potential for a flooding event downstream.

Over the last few years, there have been huge advances in scaling up high resolution weather models using state-of-the-art HPC systems making it feasible to obtain highly accurate fine-grained forecasts for large geographical regions with sufficient lead time [5], [6]. This, in turn, provides a huge opportunity to improve the quality of flood forecasts. Unfortunately, even though, there currently exist a number of overland flood routing models [7], [8], these models cannot make effective use of the fine-grained weather forecasts primarily due to computational requirements. Most of the existing flood modeling packages are designed for hydrologists to work on medium to high end desktops. These packages place a strong emphasis on being able to incorporate multiple different factors (e.g., vegetation type) into the modeling with an easy to use graphical interface so that hydrologists can manually build and experiment with small sized grids. While such approaches might work for a high resolution small sized domain or a coarse resolution domain covering a larger area, it is currently not feasible to scale these methods to domains covering large regions for which high resolution weather forecasts are available.

In this paper, we consider the weather and hydrology forecasting systems operational at the UBD|IBM Centre, Brunei Darussalam. The high resolution weather forecasting system is comprised of three levels of nested domains with the innermost domain covering approximately 180,000 square kilometers. The country of Brunei covers approximately 30% of this region. Flood models that attempt to make use of moderate resolution (90 m) Shuttle Radar Topography Mission (SRTM) [9] data for this region would require a grid size of about 2 million points while models that seek to incorporate high resolution (1m) Light Detection and Ranging (LiDAR) [10] topography data require grid sizes upwards of 15 billion points. Currently, it is not even feasible to run such models on

normal workstations due to lack of sufficient memory and large running times.

In this paper, we focus on developing scalable parallel implementations of an overland flow model on HPC systems that can be used in tandem with high resolution regional weather forecast models in an automated fashion to obtain highly accurate and precise flood forecasts with a substantial lead time (24-48 hours). We attempt to make the best possible use of what ever information is available by retaining it at the original source resolution and employing a suitable grid size, however, large.

To be specific, we make the following main contributions:

- We describe a coupled weather and flood forecasting system comprised of a high resolution regional weather model [11], [12] and an IBM proprietary hydrological model (FloodSim) based on 2D diffusive routing that can be customized for a new target region in a semi automated fashion with minimal input. To the best of our knowledge, this is the first integrated system that can be deployed for high resolution operational flood forecasting.
- We parallelize the hydrological model simulations via a nature inspired spatial decomposition of the target region into independent watershed basins, which are mapped to multiple processes using MPI. Within each process, we employ shared memory parallelization using OpenMP.
- We evaluate our implementation on two different architectures, namely an IBM Power 750 system based on the latest Power7+ processors, and an x86 system, both running Red Hat Enterprise Linux. Results indicate that we are able to achieve a speedup of up to 13× for our shared memory implementation while using 16 threads for a single large basin. Even for a test case with large number of small basins, our hybrid implementation demonstrates a speedup of up to 10.4×.

The rest of the paper is organized as follows: Section II discusses related work on parallelization approaches for flood routing engines. Section III provides an overview of our integrated weather and flood modeling framework. Details on the routing algorithm as well as distributed and shared memory parallelization approaches are presented in Section IV. Section V describes empirical evaluation of our parallelization strategies on two real world domains. Concluding remarks and directions for future work are discussed in Section VI.

II. Related Work

In this section, we briefly discuss existing work on scaling up water routing algorithms for flood modeling. Broadly speaking, existing approaches to parallelize 2D water routing algorithms can be grouped into two categories corresponding to functional and domain decomposition respectively.

Of these, the first class of methods based on functional decomposition involves parallelization of the nested loops for processing the cells of the grid. In particular, Neal et al [13], explored the intrinsic parallelism presented in the functions that looped around the floodplain cells of the domain via OpenMP. In this study, a maximum speedup of 5.8× over the serial algorithm was reported for 8 cores with domain sizes varying from 3,000 to 3 million cells. Greater speedups were observed on large domains. The key limiting factors for the parallel speedup were the serial time and processor load imbalances.

The second class of methods employ domain decomposition, where the grid to be simulated is split into smaller domains that are processed in parallel. The main challenge in this case is figuring out how to perform such a division while achieving load balance. This task is particularly difficult, since the computation costs depend not only on static properties of the domain, but also on whether there is any accumulation in the neighborhood. For instance, when half of the sub-domains all have dry cells while the other half contains wet cells, the processors responsible for the dry sub-domains will be mostly idle while the rest are busy routing water amongst the wet cells. Since the focus of our work is scaling up flood modeling to larger grid sizes, we adopt the second approach, i.e., domain decomposition in our work.

There is already a large body of literature [14], [15], [16] on using domain decomposition to improve scalability of hydrological models via message-passing interfaces (MPI) with domain decomposition. In particular, Yu [16] presents an approach to parallelize a two-dimensional diffusion-based flood inundation model wherein the target region is divided spatially into sub-regions of equal size and dimension according to the number of processors available. Several ways of spatial decomposition of the simulation domain are evaluated such as splitting it into a number of fixed columns, rows or different configurations of quadrants. Empirical performance evaluation of the proposed approach on a large domain (232000 cells) indicate a maximum speedup of 1.75×, 1.98× and 2.71× for MPI simulations using 2 nodes, 4 nodes and 8 nodes, respectively, with associated efficiencies of 0.87, 0.50 and 0.33.

Our current work extends existing work on domain decomposition via a hybrid parallelization schemes that combines watershed basin-based decomposition via MPI and intra-basin decomposition via OpenMP leading to a more efficient utilization of computing resources.

III. Integrated Flood Modeling System

In this section, we provide a brief overview of our integrated high resolution flood forecasting system. Figure 1 shows the key components of our flood modeling system and their interactions. Given a specified target region, one needs to perform a one-time region-specific domain setup to incorporate publicly available static information on the region. Post setup, there is a daily operational process which involves (a) obtaining precipitation estimates typically via a weather model, (b) determining expected surface runoff based on the precipitation estimates via a soil model, and (c) distributing the excess water via a flood routing engine to estimate flood accumulation. Additionally, at periodic intervals, the daily flood forecasts are compared with historical data as well as on-the-ground observations for verification and calibration purpose. We describe each of these above tasks and the associated software components in more detail below.

A. Static Domain Setup

Typically, there are multiple heterogeneous types of information that need to be taken into account by an overland flood model in order to represent the physical processes of any target geographical region with good fidelity. Most of this information is already captured within publicly available datasets derived from space missions [17] and field sampling [18], [19]. However, the different datasets often employ varying spatial resolutions and file formats, which makes the process of manually setting up domains for a new target region very time-consuming and error-prone. To simplify this process, our system contains a component called the *Domain Builder*, which automates obtaining and processing topography data, land use, soil type and properties, vegetation maps and water bodies, from specified data sources. Examples of such data sources include Shuttle Radar Topography Mission (SRTM) data, Moderate-Resolution Imaging Spectroradiometer (MODIS) land data products, OpenStreetMap, and Harmonized World Soil Database (HWSD). Besides its ability to pull data from public repositories, the domain builder module can also incorporate private data sources such as those produced by city survey departments. The data obtained from different sources is processed to produce a unified geo-referenced digital elevation model (DEM) annotated with all the relevant surface attributes for the target domain. This digital elevation model is then used to automatically delineate the watershed basins within a domain using various surface depression analysis techniques [20], [21], which can aid in speeding up the flood simulations as explained in the next section (Section II). The above setup process is extremely critical for the modeling processes, but is typically done only once for a target domain since the relevant information is mostly static.

B. Daily Operational Process.

As mentioned earlier, the typical operational flow is comprised of the following three steps:

Precipitation Forecasting. The quality and utility of flood predictions is highly dependent on the accuracy and timeliness of the input precipitation values. Therefore, to obtain reliable precipitation estimates, our system uses a high resolution weather forecasting service based on the Weather and Research Forecasting model (WRF), which is a state-of-the-art regional to global-scale numerical weather prediction model used by weather agencies all over the world. To obtain high prediction accuracy over smaller populated regions of interest for flood modeling purpose as well as to minimize computation for generating timely forecasts, we employ nested domain configurations with one or more carefully positioned high resolution domains (nests) embedded into a coarse resolution parent domain [12]. For instance, in Brunei and Rio de Janeiro (Brazil) where our flood forecasting is currently operational, the WRF model is set up to provide a 48-hour weather forecast at a temporal resolution of 10 minutes with a spatial resolution of 1-1.5 km (1 km in Rio and 1.5 km in Brunei). The WRF output includes accumulated total cumulus precipitation, accumulated total grid scale precipitation (micro-physics) as well as the latitude/longitude matrices of the domain covered by the forecast.

Surface Runoff Estimation. The water available for overland flow is determined by the incoming precipitation as well as the absorption properties of the soil and other relevant environmental attributes (e.g., wind speed, temperature). Though our system is primarily being deployed for operational flood forecasting, it is designed to be compatible with different soil models backends, which could be appropriate depending on the characteristics of the input data and on the aim of the simulation. In particular, for surface runoff flood simulation, we employ a soil model that focuses on the physics of water absorption by the soil and the stage in which the soil becomes saturated [22].

It is also possible to use soil models of resolution different from that of the DEM generated during the initial domain set up. In such a case, the coarse resolution surface runoff values are mapped to the fine-grained geo-referenced digital elevation model (DEM) by performing suitable cropping and interpolation. For instance, one might choose to employ the soil model provided as part of WRF with a resolution of 1km, while the DEM may have a resolution of 90m (in case of SRTM data), 30m (in case ASTER [23] data is used), or 1m (in case LiDAR data is available).

Flood Routing. The flooding status of any specific location or to be specific, the water accumulation (height) is determined by the excess water available for flow and how exactly it is routed around based on various topological characteristics. To estimate the flood accumulation, our system employs a flood routing engine that takes as input two grids of identical resolution: (a) the digital elevation model (DEM), and (b) the surface runoff estimates (SR), and performs a series of simulation steps that involve local distribution of water. The flood routing simulation is the core component of flood modeling. Currently, it is also the most compute-intensive task and needs to be optimized in order to scale up the overall flood modeling process.

As in the case of soil models, our system is designed to be compatible with different routing engines, which could be chosen depending on the requirements of the simulation and constraints arising from the grid size. Broadly speaking, there are two types of routing approaches: steady-state methods, where the output is the final solution for a certain domain, and unsteady-state algorithms wherein we can obtain a solution for intermediate time steps. For fine resolution operational flood forecasting, it is preferable to choose steady state routing since these methods scale better to grids of higher resolution. It is also possible to use the steady-state routing engine to downscale unsteady state engine output on a large coarse domain to a high resolution DEM. Section IV provides more details on the steady-state routing algorithm we employ for operational flood forecasting.

C. Verification & Calibration.

The operation of the three modeling components in the daily forecasting is determined by various configuration files specifying target region-specific values for various parameters (e.g., geographical grid nesting settings, model physics and dynamics settings). For the very initial forecasts, these parameter values are often chosen by domain experts, but have to be regularly fine-tuned based on the true ground observations in order to achieve high quality predictions. Hence, our system also comprises a verification component that automates the

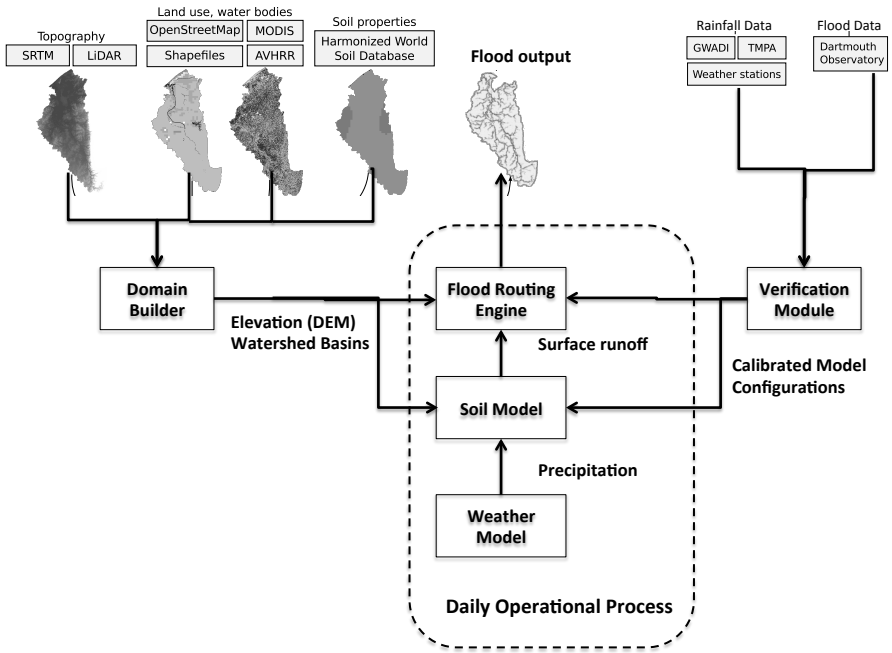

Fig. 1. Architecture of our integrated flood forecasting system.

process of obtaining actual historic rainfall and flood data from public sources such as weather stations, Dartmouth Flood Observatory [24] and evaluating the model predictions against the true observations. The results of verification can be used to calibrate the model configuration parameters.

IV. Flood Routing Engine

In this section, we describe our flood routing approach in detail. To ensure scalability, we employ a steady-state routing approach that computes the final accumulation. However, to mimic the characteristics of water flow, we have implemented the routing algorithm in terms of processing small blocks of a domain with a sliding window. Multiple iterations of this algorithm gives an estimation of the path that the water will take. The number of ideal iterations can be determined as a ratio of the maximum distance between the basins' inlets and outlets and the block size. This approach works well in practice and is currently operational in Brunei and in the city of Rio de Janeiro in Brazil to issue alerts to disaster management agencies. We first present the basic water routing algorithm in Section IV-A. Then, we describe a distributed memory (MPI) implementation based on partitioning the domain into watershed basins in Section IV-B followed by a discussion of a shared memory (OpenMP) implementation in Section IV-C.

A. Basic Water Routing Algorithm

The routing engine is provided two input grids (elevation (DEM) and surface runoff (SR)) of identical resolution and expected to return an output grid corresponding to the water accumulation at each location. To determine the final accumulation, we make use of the principle of hydrostatic equilibrium which states that fluid bodies in contact will attain the same overall height (water accumulation + natural elevation) in steady state. We apply this principle repeatedly at a local level by dividing the grid into small overlapping blocks of size $n = m \times m$ (typically $n = 9$) assuming that within each block, the fluid from any cell can flow to any other cell. If the water accumulation was large enough to cover all the cells in a block at steady state, then the overall steady state height is given by the sum of the total accumulation in the block and total natural elevation divided by the number of cells. The steady accumulation for each cell is then given by difference of the total height and the natural elevation. When the water accumulation in the block is not large enough to cover all the cells, then the water height is determined by only a subset of the cells with lower elevation, but will be uniform across those cells and the water accumulation is zero elsewhere.

The details of the algorithm are provided in Algorithm 1 and also illustrated with an example in Figure 2. The flood routing algorithm sorts grid cell in ascending order of elevation and progressively distributes water to attain equal overall height. After each block is processed, the algorithm repeats the distribution over the immediately adjacent overlapping block to ensure carry over of water accumulation in case of a gradient. To the best of our knowledge, this is the first attempt at applying this approach for flood forecasting.

B. Distributed Memory Parallelization

A key characteristic of flood modeling is that there exists a nature-insipired domain decomposition that allows efficient distributed multi-processing. To be specific, the topography of the domain typically allows one to partition the target region into sub-regions delineated by hills, mountains, and depressions. These sub-regions are called *watershed basins*

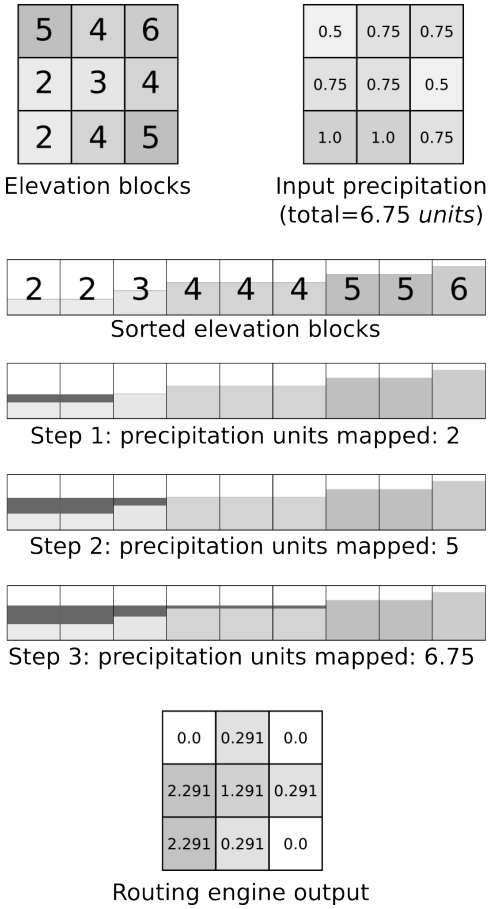

Fig. 2. Example to illustrate steady start routing algorithm

or *catchments* and there already exist a number of techniques [20] [21] to automatically identify these divisions from the digital elevation model using characteristics of the slope. Figure 3 represents the delineation of Brunei, as produced by SAGA GIS [25].

By their very nature, most of the routing computation for a given basin will rarely have to include results of the computation of a neighbor basin. So in effect, we have an embarrassingly parallel simulation at the level of watershed basins, which makes the division of workload based on watershed basins a natural choice. Communication between two neighbor basins is necessary only in situations where the water height within a basin reaches the elevation of one of its spill points, in which case a water transfer occurs. A synchronization primitive needs to be employed when such points are computed. Alternatively, one may also join nearby basins so that they form a bigger sub-region, thereby reducing the complexity of the software.

To exploit the natural domain decomposition along watershed basins, we first assign a unique identifier to each basin and the cells within them, which can be used a mask to determine assignment to computing resources. We adopt a master-slave approach for distributed processing as shown in Algorithm 2. The master node maintains a list of all the basins in the domain sorted by their size in descending order as well as all the relevant details. It is also responsible for assignment of basins

Algorithm 1 Routing engine

Routine: Distribute_water()

Input:
$C_i.h$: the topographical height of ith cell in the block, $\forall[i]_1^n$
$C_i.x$: the water accumulation of ith cell in the block, $\forall[i]_1^n$
Output:
$C_i.x$: Updated water accumulation of ith cell, $\forall[i]_1^n$

{Initialize remaining accumulation}
$RA \leftarrow C_1.x + C_2.x + \ldots + C_n.x$
{Reset cellwise accumulation}
$C_i.x \leftarrow 0, \ \ \forall[i]_1^n$
{Sort cells based on elevation in ascending order}
$s \leftarrow sort([C_1.h, \ldots, C_i.h, \ldots, C_n.h])$
{Initialize water level} $currH \leftarrow C_{s(1)}.h$
{Initialize end position} $endPos \leftarrow 2$
while (RA $>$ 0 && endPos $\leq$ n)
{Identify cells to distribute water }
while ($currH == C_{s(endPos)}.h$)
endPos++
end while
numCells $\leftarrow$ endPos-1
{Compute difference in height and possible redistribution}
$diffH \leftarrow C_{s(endPos)}.h - currH$
$extraH \leftarrow min(RA/numCells, diffH)$
{Redistribute water to selected cells and compute new height}
$C_i.x \leftarrow C_i.x + extraH \ \ \forall i, \ s(i) < endPos$
$currH \leftarrow currH + extraH$
{Recompute remaining accumulation}
$RA \leftarrow RA - numCells * extraH$
end while
{ Distribute the leftout water to all cells}
if (RA $>$ 0)
$C_i.x \leftarrow C_i.x + RA/n, \ \ \forall[i]_1^n$
end if

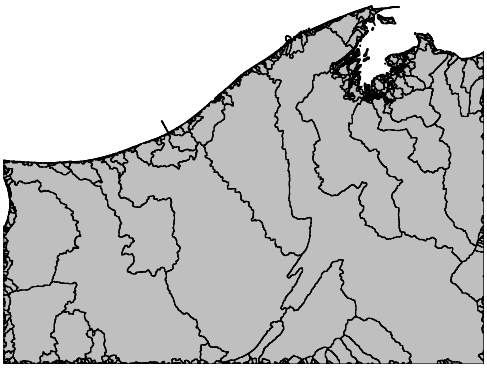

Fig. 3. Delineation of Brunei area into 1358 watersheds

to the worker nodes and for aggregating the output of all the worker nodes.

The basins are assigned and processed as follows. Idle worker nodes send short messages to the master indicating their availability as well as the maximum basin size that they can currently process. This maximum basin size is a heuristic estimate based on the relative ratio of the product of the number of cells in the basin and the number of attributes that

need to be kept in memory versus the available memory on that node. Such a size-based selection criterion is especially useful when spreading the workload across a heterogeneous cluster. On receiving the worker request, the master answers it with an ACK followed by description of a basin smaller than the size requested by the worker node or a NACK in case there is no such basin available indicating that the worker can shut down. The cost of transferring of this basin description is negligible. The worker node processes the assigned basin using the water distribution algorithm described in Section IV. On completion, it writes out the simulation output and signals its availability to the master node. Once all the basins are processed, the master node aggregates all the partial output files, producing a series of images representing the water accumulated on the surface after each simulation step.

Algorithm 2 Distributed processing

Trigger– *Simulation Starts : In MasterNode*

Create a list of all basins sorted in descending order of size
while list of unassigned basins $\neq \emptyset$
Receive request (with maxBasinSize) from idle node n_i
Identify basin B for node n_i based on maxBasinSize
if (B $\neq$ NULL)
Send ACK and details of basin B to the node n_i.
else
Send NACK to node n_i
end if
end while
Wait for workers to complete
Aggregate the worker output grids

Trigger– *Simulation Starts : In WorkerNode*

Compute maxBasinSize based on available memory
Send a request for basin with maxBasinSize
Receives ACK with basin details or NACK
while master response is ACK
Set up the grid using the DEM for the basin.
Initialize the accumulation for each grid cell to 0.
for each time instant,
Update cell accumulation with surface-runoff.
Scan grid row-wise, do **Distribute_water()** for every cell.
end for
Send the output of basin simulation to master node
Compute maxBasinSize based on available memory
Send a new request for basin with maxBasinSize
Receives ACK with basin details or NACK
end while

C. Shared Memory Parallelization

A very large fraction of the total simulation time is spent on routing water amongst the grid point within watershed basins. For instance, the routing time for the domain with basin size 9456x7390 contributed to nearly 84% of the total simulation time. As the time intervals get shorter, the computational effort for routing becomes even more dominant. Therefore, in addition to the watershed basin based parallelization across multiple nodes described in Section IV-B, we also consider a shared memory based parallelization of a single watershed basin across multiple OpenMP threads.

The key idea in such a shared memory parallelization approach is to partition the domain into disjoint sub-areas such that each sub-area is independently processed by a separate OpenMP thread. In order to achieve effective parallelization of the flood routing simulation, we need to address two main challenges:

- **Partitioning irregular shaped domains.** Watershed basins typically tend to be highly irregular in shape as shown in Figure 4. Direct domain decomposition of an irregular grid into possibly irregular sub-regions is non-trivial. On the other hand, mapping it to a regular grid structure results in grid cells that are not required for simulation, in turn leading to wasted computation, and additional book-keeping. Further, partitioning the resulting regular grid into sub-areas based on grid points alone might result in heavy load imbalance among the threads.
- **Handling race conditions.** Another important cause for concern is that even though the tiles are all processed independently, the parallel processing of the shared domain introduces race conditions at the common tile boundaries. It is, therefore, important to ensure that threads responsible for processing neighboring tiles have mutually exclusive access to the boundary cells.

In our current work, we adopt a relatively straightforward approach to address the above two issues by mapping the irregular watershed basins to rectangular grids, partitioning each rectangular grid into rectangular tiles (possibly of different tile size, but nearly equal workload) one for each thread, and adopting a quadrant-based synchronization to ensure no conflicts among threads processing neighboring tiles. We discuss the main elements of our approach in more detail below.

Domain Embedding and Tile Layout. For each watershed basin, we consider the bounding rectangular grid (see Figure 4) as the shared domain to parallelize. This rectangular domain is then divided into rectangular tiles based on the intended degree of parallelization such that each thread gets a single tile. The tiles are constructed following a two level splitting process: first into vertical slices with each vertical slice further split into an equal number of horizontal ones. Let N be the number of threads and hence, the desired number of tiles. Then, the number of horizontal and vertical partitions (N_x and N_y respectively) have to be chosen such that $N_x N_y = N$. The factors are picked so that $N_x \simeq N_y$. Lastly, to ensure a balanced aspect ratio for each individual tile, the factors N_x and N_y are assigned to dimensions so that the larger dimension of the rectangular domain has more partitions. For instance, for a domain of size 1200×500 with 32 threads, we pick a 8×4 tile layout since that is integral factorization with the closest factors (compared to other possibilities (1,32), (2,16)), and assign the larger number of partitions to the longer dimension (i.e. 8 to 1200).

Balancing Tile Workloads. Typically, given the tile layout $N_x \times N_y$, partitioning a rectangular domain into tiles is trivial. However, as pointed out earlier, only a small fraction of

grid points of the original rectangular domain are required for simulating the original watershed basin due to which the trivial decomposition leads to load imbalance. To achieve load balancing, we explicitly keep track of the *valid grid points*, which include any point in the original watershed basin with a valid height value (considering the possibility of some missing values). The tiles are constructed so as to have nearly equal number of valid grid points.

Let $m_x \times m_y$ be the size of the rectangular grid. Let $C(i,j)$ denote the number of valid grid points in the sub-grid with corners $[(0,0),(i,0),(0,j),(i,j)]$ inclusive of the boundaries. The total number of valid grid points in the grid is thus $C_{tot} = C(m_x, m_y)$ and valid grid points up to and including the ith column is $C(i, m_y)$. Since the tile construction involves a vertical slicing followed by a horizontal slicing of each of the vertical slices, given a tile layout $N_x \times N_y$, we need to determine the vertical boundaries $\{p_x(g),\ [g]_1^{N_x}\}$ as well as the horizontal boundaries $\{p_y(h,g)\ [h]_1^{N_y},\ [g]_1^{N_x}\}$.

To ensure nearly equal distribution of valid points, the gth vertical boundary $p_x(g)$ is picked so that it is the smallest column index such that the number of valid grid points over all rows and up to the $p_x(g)^{th}$ column exceeds $\frac{gC_{tot}}{N_x}$, i.e.,

$$C(p_x(g)-1, m_y) < \frac{gC_{tot}}{N_x} \leq C(p_x(g), m_y),\ [g]_1^{N_x}.$$

Similarly, for each vertical slice (say the gth one), we pick the hth horizontal boundary $p_y(g,h)$ such that it is the smallest row index such that the number of valid grid points on all the columns in the slice and up to the $p_y(g,h)^{th}$ row exceeds $\frac{h}{N_y}$ fraction of the total valid points in the slice.

$$\begin{aligned} &C(p_x(g), p_y(g,h)-1) - C(p_x(g-1), p_y(g,h)-1) \\ &\quad < \frac{h}{N_x}\left(C(p_x(g), m_y) - C(p_x(g-1), m_y)\right) \\ &\leq C(p_x(g), p_y(g,h)) - C(p_x(g-1), p_y(g,h))\ [g]_1^{N_x}, [h]_1^{N_y}. \end{aligned} \tag{1}$$

We implement the above partitioning efficiently by storing and reusing the results of prior computations. Fig 5 shows the tiles formed for 8 threads over a watershed basin of resolution 9456x7389 with 35234396 total valid points and a 4x2 tile based partitioning. Table I summarizes the valid cells in each of these tile. As one can see, the tiles have nearly equal number of valid points.

Quadrant-based Synchronization. Lastly, to handle race conditions, we divide each tile into quadrants, or in other words, even smaller tiles following a 2x2 layout. Due to the rectangular shape of the tiles, there can be at most four neighbors that have common boundary grid points with the boundary point belonging to a different quadrant in each of the neighbors. Given a specific quadrant position (1 out of 4 possible choices), there are no shared boundaries between the corresponding quadrants of the different tiles. Hence, we treat the quadrants as the basic unit of work for each thread and process the four quadrants following a strict order with the execution of each quadrant synchronized amongst threads by a barrier to ensure work isolation. Since there is synchronization after each quadrant processing, it is also important to ensure that the workload of a specific quadrant across the tiles is nearly equal. As the tiles are already nearly equal in the number of valid points, we achieve quadrant-level load balance by splitting the tile into quadrants with nearly equal number of valid points. This splitting procedure of tiles into quadrants (sub-tiles) is done in exactly the same manner as the splitting of the 2D grid into tiles following a 2 x 2 layout (i.e., a single horizontal slice with each slice having a single vertical split). Note that as a side effect of parallelization, we may read obsolete values of neighboring grid points for processing boundary points, but there are negligible effects due to adjustments over multiple passes. Table I shows the load-balanced quadrants within each of the 8 tiles corresponding to the 8 threads for the watershed basin (9456x7389) referenced earlier.

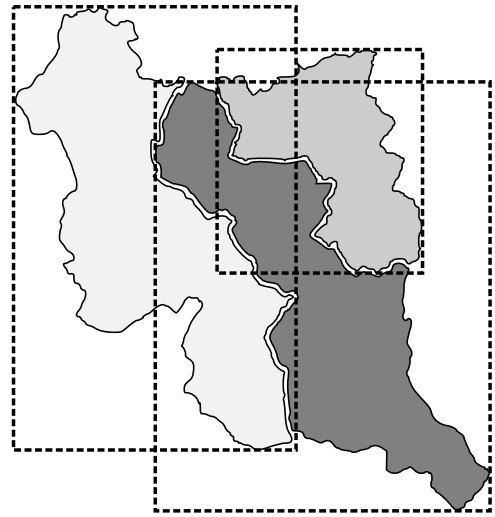

Fig. 4. Watershed domains delimited by bounding boxes

TABLE I. LOAD BALANCING W.R.T. VALID CELLS WITH 8 THREADS FOR WATERSHED GRID OF RESOLUTION 9456X7389 AND 35234396 VALID POINTS

Tile	Quadrant 1	Quadrant 2	Quadrant 3	Quadrant 4	Total Cells
T1	1103017	1101667	1101305	1099840	4405829
T2	1101628	1101660	1100930	1099602	4403820
T3	1102112	1101258	1102001	1100100	4405471
T4	1102209	1101374	1101050	1100593	4405226
T5	1101965	1101492	1101022	1101492	4405971
T6	1101492	1101342	1101492	1099158	4403484
T7	1101881	1101030	1101705	1099184	4403800
T8	1101000	1100294	1100519	1098982	4400795

V. EMPIRICAL EVALUATION

In this section, we describe experimental evaluation of the proposed parallelization strategies on two real-world domains.

A. Empirical Setup

Hardware Configuration. For our experiments, we used two different hardware configurations. The first configuration is a Power 750 express server consisting of 32 Power7+ processor cores, running Red Hat Enterprise Linux version 6.4. Each 3.3 GHz core has a 256 KB L2 cache and a 10 MB L3 cache with 256 GB of total RAM. The second configuration is a Dell Precision T7600 system having 8 core Intel Xeon processors running 64 bit Ubuntu 12.04 LTS. This machine has a higher

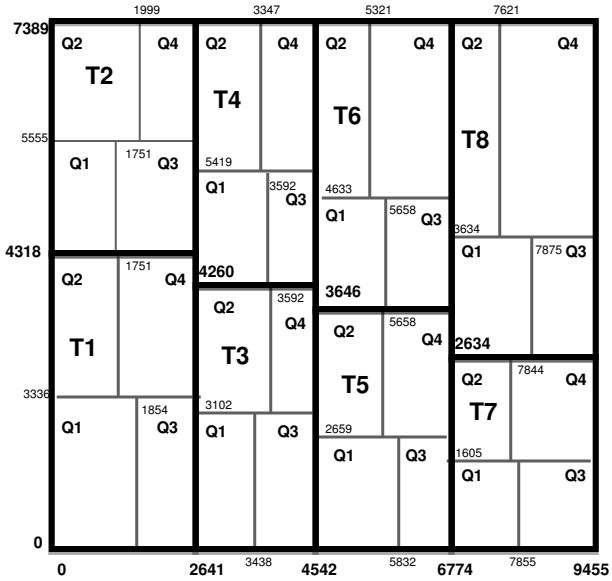

Fig. 5. Tile and quadrant based partitioning with 8 threads for watershed basin of resolution 9456x7389. The thick lines show the tile boundaries while thinner lines show the quadrant boundaries. The numbers represent the coordinates of tiles and quadrants(bold numbers representing the tile coordinates).

20MB cache for all the 2GHz processor cores, supports hyper-threading, and has 256 GB DDR3 RAM. Both these machines provide uniform memory access and hence memory access performance is ignored in this study. We primarily considered these configurations since a large fraction of high performance computing systems are based on Intel and Power architectures. The flood modeling software was compiled using the same compilers on both the systems. Specifically, we used the GNU compiler collection with -O3 optimization since these are two very different architectures and have dedicated compiler backends that exploit the hardware features better.

Domains. We consider two real world domains with different characteristics.
Brunei-MPI. The operational flood model in Brunei uses a DEM of size 1688x1318 derived from global 90m SRTM data. This domain has 1358 watershed basins as shown in Figure 3. We primarily use this domain to show the benefits of our MPI based parallelization of the basins. For all simulations using this domain, we used the precipitation forecast from a hindcast of the heavy flooding event in January, 2009. We also test our hybrid MPI/OpenMP implementation on this domain.

Rio de Janeiro-OpenMP. We extracted a large basin of size 9456×7390 from the 1m resolution LiDAR data for the city of Rio de Janeiro. In early April 2010, Rio endured the worst rainstorms in 48 years, which was considered one of the most significant, global weather events of 2010. We performed a hindcast using our high resolution weather model and obtained estimates of precipitation which were used as input to the flood model.

B. Scaling of MPI Implementation

Figures 6 and 7 show the scaling of our distributed memory implementation on the Power and Intel machines on the Brunei domain with 1358 watersheds. We observe that the speedup flattens for 8 and more processors. This is due to the fact that there are 4 large basins that occupy about 19%, 14%, 12.5%, and 7.5% of the total area covered by the domain. Towards the end of the simulations, these 4 basins are a major cause of load imbalance since all but four processors are idling. Hence, we observe a drop in speedups for 8 and more processors.

C. Scaling of OpenMP implementation

Figure 8 shows the speedups obtained for our OpenMP based parallelization on the Rio de Janeiro domain. Since the OpenMP parallelization is applicable only for the route water part, we also show the speedups obtained for the routing part alone. We obtain very good scaling of up to $13\times$ on 16 threads for our threaded implementation on the Intel machine. However, the reduced speedups for the total time taken for this

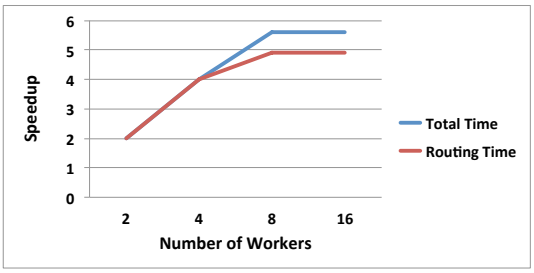

Fig. 6. MPI: Speedup on Power for Brunei domain.

basin suggest that the sequential components are becoming a bigger fraction of the total time. We observe similar scaling behavior on the Power machine.

D. Scaling of Hybrid Implementation

We also compare the speedup of hybrid implementation on both the hardware configurations using a maximum value of 16 for $workers \times threads$ combination since this was the maximum number of threads available on the Intel machine. Tables II and III show the speed up achieved on the Intel machine for total and routing times respectively. We had already noticed that beyond 8 workers, the maximum speedup that can be obtained for this domain is 6.5 pointing to diminishing returns for the single thread case. However, with the hybrid version, we are able to extract a speedup of up to 10.4× for the total time.

However, when one considers just the routing time, we observe a maximum speedup of 10.6× obtained for the combination of 4 workers and 4 threads. This is mainly due to the combined effect of higher speedups for the OpenMP enabled sections and the small number of moderate sized basins for this test case. Initial results indicate that our hybrid approach will do well in situations with higher number of large sized basins. The best combination of workers and threads will be a function of the total number of basins, average size of large basins as well as the number of large basins.

Tables IV and V show similar behavior for the Power machine. We note that there is a slight difference in the speedups for the two machines. This difference can be attributed to the fact that the configurations for both these systems are different and we have not explored all possible architecture specific fine-tuning for the respective systems.

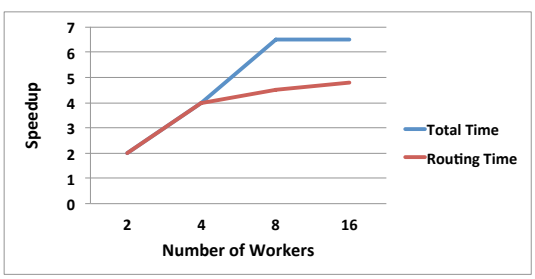

Fig. 7. MPI: Speedup on Intel Xeon for Brunei domain.

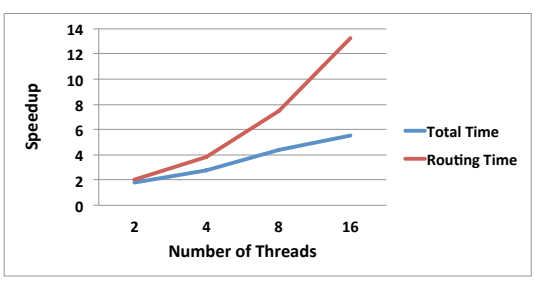

Fig. 8. OpenMP: Speedup on Intel Xeon for Rio de Janeiro domain.

TABLE II. SPEEDUP W.R.T. TOTAL TIME ON INTEL XEON FOR BRUNEI DOMAIN.

	1 thread	2 threads	4 threads	8 threads	16 threads
1 Worker	1	1.5	2.2	2.5	3.1
2 Workers	2	3.3	3.7	5.2	–
4 Workers	4	5.8	8.7	–	–
8 Workers	6.5	10.4	–	–	–
16 Workers	6.5	–	–	–	–

TABLE III. SPEED UP W.R.T. ROUTING TIME ON INTEL XEON FOR BRUNEI DOMAIN.

	1 thread	2 threads	4 threads	8 threads	16 threads
1 Worker	1	1.7	3.2	4.2	7.6
2 Workers	2	3.8	4.9	9.4	–
4 Workers	4	6.5	10.6	–	–
8 Workers	4.5	8.3	–	–	–
16 Workers	4.8	–	–	–	–

VI. CONCLUSION & FUTURE WORK

Operational flood forecasting is an extremely important problem requiring highly scalable high resolution integrated modeling approaches. Our current work presents such an integrated modeling system comprised of weather models, soil models and routing engine. In particular, we focus on the routing process which is the most compute intensive and propose parallelization strategies to scale it up to large grid sizes. First, we make use of naturally occurring parallelism in flood routing via a master slave model of parallelization for distributed processing of the basins. We also propose an intra-basin shared memory parallelization approach using OpenMP. Empirical evaluation of the proposed parallelization strategies indicates a potential for high speedups for certain types of scenarios (e.g., speedup of 13× with 16 threads using OpenMP parallelization for the large Rio de Janeiro basin).

In future, we plan to develop a completely distributed memory approach even for intra-basin parallelization, which would be essential in order to use LiDAR data based DEM (which is much more fine-grained than the current coarse SRTM based topography). Such an approach will also permit the routing engine to be ported to large scale supercomputers such as the Blue Gene P/Q. Another possible direction of exploration is a hybrid approach that combines inter-basin MPI/OpenMP, intra-basin MPI/OpenMP in a single simulation depending on the characteristics of the domain and the constituent basins. Lastly, there is also scope for optimizing the basic water routing algorithm itself by an intelligent exploration of the subsets of the cells with non-zero accumulation.

TABLE IV. SPEED UP W.R.T. TOTAL TIME ON POWER FOR BRUNEI DOMAIN.

	1 thread	2 threads	4 threads	8 threads	16 threads
1 Worker	1	1.4	1.8	2.7	3.4
2 Workers	2	2.7	3.5	5.6	–
4 Workers	4	6.3	7.8	–	–
8 Workers	5.6	9.2	–	–	–
16 Workers	5.6	–	–	–	–

TABLE V. SPEED UP W.R.T. ROUTING TIME ON INTEL XEON FOR BRUNEI DOMAIN.

	1 thread	2 threads	4 threads	8 threads	16 threads
1 Worker	1	1.5	2.0	3.6	5.0
2 Workers	2	2.9	3.9	7.3	–
4 Workers	4	6.5	8.7	–	–
8 Workers	4.9	8.1	–	–	–
16 Workers	4.9	–	–	–	–

ACKNOWLEDGMENT

The authors would like to thank the city of Rio de Janeiro and the Instituto Pereira Passos for the availability of the high resolution dataset used for the experiments shown in this paper. We would also like to thank UBD|IBM Centre, Universiti Brunei Darussalam for providing access to the weather forecasting service and the Intel machine used for this work.

REFERENCES

[1] "Natural catastrophes worldwide : Percentage distribution for year 2012." [Online]. Available: http://www.munichre.com/app_pages/www/res/pdf/NatCatService

[2] M. Pitt, *Learning lessons from the 2007 floods : An independent review*. [Online]. Available: http://www.cabinetoffice.gov.uk/thepittreview.aspx.

[3] W. C. S. et al., "A description of the Advanced Research WRF version 3," *NCAR Technical Note TN-475*, 2008.

[4] C. Vreugdenhil, *Numerical Methods for Shallow-Water Flow*, ser. NATO Asi Series. Series C, Mathematical and Physical Science. Springer, 1994.

[5] J. M. et al., "WRF Nature Run," in *SC*, 2007.

[6] P. Malakar, T. George, S. Kumar, R. Mittal, V. Natarajan, Y. Sabharwal, V. Saxena, and S. S. Vadhiyar, "A divide and conquer strategy for scaling weather simulations with multiple regions of interest," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 37:1–37:11.

[7] E. Todini, "The ARNO rainfallrunoff model," *Journal of Hydrology*, vol. 175, no. 14, pp. 339 – 382, 1996.

[8] M. A. Gill, "Flood routing by the Muskingum method," *Journal of Hydrology*, vol. 36, no. 34, pp. 353 – 363, 1978.

[9] J. A., H. Reuter, A. Nelson, and E. Guevara, "Hole-filled SRTM for the globe, version 4," 2008. [Online]. Available: CGIAR-CSI SRTM 90m Database (http://srtm.csi.cgiar.org).

[10] G. Priestnall, J. Jaafar, and A. Duncan, "Extracting urban features from LiDAR digital surface models," *Computers, Environment and Urban Systems*, vol. 24, no. 2, pp. 65 – 78, 2000.

[11] L. A. Treinish, A. P. Praino, J. P. Cipriani, U. T. Mello, K. Mantripragada, L. C. V. Real, P. A. Sesini, V. Saxena, T. George, and R. Mittal, "Enabling high-resolution forecasting of severe weather and flooding events in Rio de Janeiro," *IBM Journal of Research and Development*, vol. 57, no. 5, 2013.

[12] P. Malakar, V. Saxena, T. George, R. Mittal, S. Kumar, A. Naim, and S. A. b. H. Husain, "Performance evaluation and optimization of nested high resolution weather simulations," in *Euro-Par 2012 Parallel Processing*, ser. Lecture Notes in Computer Science, C. Kaklamanis, T. Papatheodorou, and P. Spirakis, Eds. Springer Berlin Heidelberg, 2012, vol. 7484, pp. 805–817.

[13] J. Neal, T. Fewtrell, and M. Trigg, "Parallelisation of storage cell flood models using OpenMP," *Environmental Modelling & Software*, vol. 24, no. 7, pp. 872 – 877, 2009.

[14] B. F. Sanders, J. E. Schubert, and R. L. Detwiler, "ParBreZo: A parallel, unstructured grid, Godunov-type, shallow-water code for high-resolution flood inundation modeling at the regional scale," *Advances in Water Resources*, vol. 33, no. 12, pp. 1456 – 1467, 2010.

[15] J. C. Neal, T. J. Fewtrell, P. D. Bates, and N. G. Wright, "A comparison of three parallelisation methods for 2D flood inundation models," *Environ. Model. Softw.*, vol. 25, no. 4, pp. 398–411, Apr. 2010.

[16] D. Yu, "Parallelization of a two-dimensional flood inundation model based on domain decomposition," *Environmental Modelling & Software*, vol. 25, no. 8, pp. 935 – 945, 2010.

[17] S. Zhao, W. Cheng, C. Zhou, X. Chen, S. Zhang, Z. Zhou, H. Liu, and H. Chai, "Accuracy assessment of the ASTER GDEM and SRTM3 DEM: an example in the Loess Plateau and North China Plain of China," *Int. J. Remote Sens.*, vol. 32, no. 23, pp. 8081–8093, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1080/01431161.2010.532176

[18] F. Nachtergaele and N. Batjes, *Harmonized World Soil Database*. FAO/IIASA/ISRIC/ISSCAS/JRC, 2012.

[19] T. Kawanishi, H. Kuroiwa, M. Kojima, K. Oikawa, T. Kozu, H. Kumagai, K. Okamoto, M. Okumura, H. Nakatsuka, and K. Nishikawa, "TRMM precipitation radar," *Advances in Space Research*, vol. 25, no. 5, pp. 969–972, 2000.

[20] L. Wang and H. Liu, "An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling," *International Journal of Geographical Information Science*, vol. 20, no. 2, pp. 193–213, 2006.

[21] X. Chu, J. Zhang, Y. Chi, and J. Yang, "An improved method for watershed delineation and computation of surface depression storage," in *Watershed Management 2010: Innovations in Watershed Management Under Land Use and Climate Change*. Madison, Wisconsin, USA: American Society Of Civil Engineers, 2010, pp. 1113–1122.

[22] Y. Ma, S. Feng, D. Su, G. Gao, and Z. Huo, "Modeling water infiltration in a large layered soil column with a modified Green-Ampt model and HYDRUS-1D," *Comput. Electron. Agric.*, vol. 71, pp. S40–S47, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.compag.2009.07.006

[23] M. Abrams, "The Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER): data products for the high spatial resolution imager on NASA's Terra platform," *International Journal of Remote sensing*, vol. 21, no. 5, pp. 847–859, 2000.

[24] G. R. Brakenridge and D. Karnes, "The Dartmouth Flood Observatory: an electronic research tool and electronic archive for investigations of extreme flood events," in *Geoscience Information Society Proceedings*, 1996.

[25] SAGA Development Team, *System for Automated Geoscientific Analyses (SAGA GIS)*, Germany, 2008. [Online]. Available: http://www.saga-gis.org/