Simulating Watersheds Using Loosely Integrated Model Components: Evaluation of Computational Scaling Using OpenMI

Anthony M. Castronova[a], Jonathan L. Goodall[a,*]

[a] *Department of Civil and Environmental Engineering, University of South Carolina, 300 Main Street, Columbia,SC 29208*

Abstract

Complicated research and management questions regarding watershed systems often require the use of more than one simulation model. Therefore, it is necessary to develop a means to integrate multiple simulation models to predict holistic system response. In this paper we explore the use of a component-based approach for the runtime integration of models, implemented as "plug-and-play" software components. The motivation for this work is to quantify performance overhead costs introduced by adopting a component-based paradigm for loosely integrating hydrologic simulation models. We construct a standard rainfall/runoff watershed model using the Open Modeling Interface (OpenMI) Software Development Kit (SDK) where infiltration, surface runoff, and channel routing processes are each implemented as independent model components. We then analyze the performance of this loosely integrated model to quantify computational scaling, using the Hydrologic Engineering Center's Hydrologic Modeling System (HMS) for comparison. Our results suggest that the overhead introduced by runtime communication of data is not significant when applied for semi-distributed watershed modeling. Our analysis was limited to semi-distributed watershed modeling, however, and future research is needed to understand performance and accuracy for more data demanding hydrologic

*Corresponding author
Email addresses: Castrona@email.sc.edu (Anthony M. Castronova), Goodall@cec.sc.edu (Jonathan L. Goodall)

models.

Keywords: Hydrologic Modeling, Software Architecture, Component-based Modeling, Integrated Modeling, OpenMI

1. Introduction

Hydrologic models typically focus on a small subset of processes within the overall hydrologic cycle (Singh and Woolhiser, 2002). This approach is adequate if one assumes that the boundary conditions of the problem do not change, and are not changed by, processes happening outside of the model's scope. There is growing acknowledgment, however, that in many cases this is not a correct assumption. Groundwater and surface water provide one example. They are often modeled as separate entities, despite the fact that in many situations modeling water in one domain necessitates modeling water in the other domain as well. This example implies that the coupling between hydrologic subsystems is important. Therefore, we need to consider how to build models that are able to simulate these dynamic system interactions.

One solution to this problem is to simply extend existing models so that they include a larger scope of process representations. This approach has been taken by a number of researchers that have integrated groundwater, surface water, climate, weather, economic, and other models by altering source code so that they are able to exchange boundary conditions during a model simulation (e.g. Ahrends et al., 2008; Anderson et al., 2002; Kunstmann et al., 2008; Yu et al., 2006; Barthel et al., 2008; Mölders and Rühaak, 2002; Maxwell et al., 2007). The general approach taken in these examples can be classified as tight integration (Sui and Maggio, 1999), where components of a system are compiled into a single application. The primary advantage of this approach is that the modeler has full control over all system components, therefore computational performance can be optimized across the entire system.

While there are clear advantages to using a tight integration approach for coupling models, there are also limitations. First, in the case that a model is extended by including new code, adopting a tight integration approach increases code redundancy through a process called "model creep." Model creep is where a model originally intended for simulating one set of processes has been extended to include other processes that were previously considered predetermined boundary conditions. This is problematic because the development team are likely experts in the field that the model was originally designed to simulate, not in these related topics. The second case is that two existing models are coupled using a tight integration approach. This requires changes to the model's source code, which make it out of sync with core code development efforts. As a result, it is difficult to maintain a state-of-the-art modeling system with such an approach.

An alternative method for coupling models is to follow a loose integration paradigm that focuses on the decomposition of a system into a set of interlinked computational components (Argent et al., 2006). The simplest form of loose integration is communication through a common file format or by data conversions between model-specific file formats. This simple form of loose integration can be computationally inefficient if models require runtime communication of data. To address this limitation, loose integration can also be achieved by creating models as software libraries instead of stand-alone executables which can then be integrated into a modeling framework (Löwy, 2005). Furthermore, if the model libraries follow a standard interface specification, then it is possible to create a generic modeling framework that facilitates model-to-model data communication during simulation.

Although tightly integrated approaches for modeling hydrologic systems are likely more common, loose integration is becoming more widely used. Several frameworks have been recently developed that offer the option of joining models together in a loosely integrated, plug-and-play manner (e.g., the Community Surface Dynamics Modeling System (CSDMS), the Earth System Modeling Framework (ESMF), the Open Modeling Interface (OpenMI), etc.). These loosely integrated architectures give modelers the freedom to interchange process representations, and allows them to construct the most representative model possible for a specific system and a specific objective (Morton et al., 2003). Furthermore, these systems enable modelers to combine the contributions of legacy models with new process-level models, providing a full suite of tools to use during model development.

Despite the growing popularity of loosely integrated frameworks for modeling environmental systems, there are few studies that quantify the computational overhead introduced by the approach. Loose integration has the potential to introduce performance overheads that may significantly impact model runtime. For example, consider the Open Modeling Interface (OpenMI) version 1.4 and its mechanism for data transfer between components. In an OpenMI simulation, data is transferred between components during runtime using a request and reply paradigm (Gregersen et al., 2007). Since this can result in thousands or millions of transactions during a model simulation, we hypothesize that it may have a considerable effect on execution time. For this reason, it is expected that model communication will be a source of simulation inefficiency, which will be amplified as the number of time steps or the size of data transfers increase. We do not know if the overhead of this communication protocol makes the approach unattractive for modeling hydrologic systems. This analysis is meant to provide further insight into this issue.

Given this motivation, we investigated the performance of the OpenMI (version 1.4) for simulating rainfall/runoff using typical hydrologic engineering calculations. We used the Hydrologic Engineering Center's Hydrologic Modeling System (HMS), an industry standard model widely used in engineering practice, to verify our calculations and to provide a point of comparison for the computational scaling results. We created a rainfall/runoff model using the OpenMI and HMS for the same watershed, to simulate the same storm event, using the same hydrologic process representations. Two metrics were used to evaluate computational performance: an endurance test in which the number of time steps was increased and a load test where the number of modeling elements was increased. The goal of these experiments was to quantify how the OpenMI scales as the number and the size (i.e., memory) of communications between model components increase.

In the following section we provide further background on the distinction between tight and loose model integration techniques. Additionally, a brief introduction to the HMS and OpenMI systems is provided to familiarize the reader with key concepts. We then describe the methodology of this study, including the implementation of each model and the tests that were conducted to measure computational scaling. Finally, we present and discuss the results, and conclude with our findings and suggestions for future research.

2. Background

There are distinct differences between the loose integration and tight integration approaches. In the tight integration approach (Figure 1), the Model Coupler class directly calls routines defined internally by Models 1 and 2. These routines must be compiled into the modeling system prior to a model simulation and cannot be discovered at runtime. In contrast, the loose integration approach (Figure 2) features a Model Engine class that calls Routines 1 and 2 by instantiating an object which implements a standard interface. This standard interface, denoted as ICoupler, defines simulation methods that must be implemented in each Routine. Using this approach the Model Engine class receives data through the standard interface, having no knowledge of how it is created (Holzworth et al., 2010). This is significant because additional Routines can be added to the system and the Model Engine automatically knows how to communicate with them. Hence the system can be easily reconfigured and extended using a “plug-and-play” approach.

This “plug-and-play” characteristic of loosely integrated systems encourages modelers to add, remove, or replace components within a model configuration (Govindaraju et al., 2006). Additionally, computational redundancy is avoided by designing components to address specific parts of the overall system. This characteristic is useful in forming unique representations of complex systems, including multi-disciplinary applications, because components can be built by independent groups (Kennen et al., 2008). Loosely integrated, process-level modeling offers greater flexibility in model design as well as transparency in model execution. Using this concept, a hydrologic system can be decomposed into independent process-level components (infiltration, surface runoff, channel routine, etc.) that communicate with each other through standard interfaces (Argent et al., 2006).

In our analysis we have elected to use the Hydrologic Modeling System (HMS), developed by the US Army Corps of Engineers (USACE), as a point of comparison. HMS is a deterministic watershed model that focuses primarily on water quantity simulation (Scharffenberg and Fleming, 2009). It was selected for this study because it is widely used in hydrologic engineering analysis for simulating rainfall/runoff (e.g. Al-Abed et al., 2005; Knebl et al., 2005). Built from its predecessor HEC-1, HMS simulates hydrologic systems as an inter-connected network of hydrologic and hydraulic units (i.e. subbasins, reaches, reservoirs, junctions, etc.) (Chu and Steinman, 2009). Each of which has a unique combination of process representations and parameter definitions, that allow the user to customize the model for a specific watershed study.

An HMS model is divided into three categories: watershed physical description, meteorological modeling, and hydrologic simulation (Scharffenberg and Fleming, 2009). The physical description of a watershed includes a suite of computational algorithms to simulate infiltration losses, transformation of excess precipitation, representation of baseflow, and computation of flow in open channels (Feldman, 2000). The meteorological model contains algorithms for estimating historic and synthetic precipitation, evapotranspiration, and snow melt. Lastly, hydrologic simulation is managed by user-defined control specifications, which include the start time, end time, and computational interval (Scharffenberg and Fleming, 2009).

We elected to use the OpenMI version 1.4 as the loosely integrated framework because it is tailored to the water resources domain. It also allows for the representation of discrete geometric features such as the subbasins used in HMS. The OpenMI was developed in 2005 and was funded through the European Union

Water Framework Directive. Its design focuses on establishing an interface standard to enable model interoperability (Moore et al., 2005). As a result, it is not meant to be a complete modeling framework, but rather a standard message passing protocol that is implemented through standard interfaces (Gregersen et al., 2007). The OpenMI Association Technical Committee (OATC) has also developed a Software Development Toolkit (SDK) that extends the OpenMI standard into a modeling framework. We use this SDK along with extensions designed to more easily create process-level OpenMI components (Castronova and Goodall, 2010) to implement the loosely integrated watershed simulation.

The OpenMI consists of interface standards that are implemented by software classes to form the foundation of an integrated model. Four key concepts of this standard are linkable components, exchange items, links, and model compositions (Brinkman et al., 2005). Linkable components are objects that perform calculations and are able to accept input exchange items and produce output exchange items. Exchange items define the messages that pass between linkable components during a simulation, and consist of objects that describe the quantity and element set for a data exchange (Gregersen et al., 2007). Lastly, links connect components together by specifying the pathway through which data flows from one component the next. Once links between model components have been established, the system is referred to as a model composition and is executed with either the command line or GUI software application provided as part of the OpenMI SDK.

3. Methodology

To quantify the computational performance of the OpenMI, we modeled the same watershed system using the same mathematical relationships, but using a loosely integrated approach in one model (the OpenMI version) and using an industry standard model that follows a tight integration approach in the other (the HMS version). Our goal was to first verify that both models produced the same results, and second to quantify how each model scaled as the modeling demands increased. To do this, we conducted two tests. In the first, called the load test, each model executes multiple simulations in which the number of computational elements is gradually increased to an upper limit. The second, named the endurance test, evaluates how each model responds to an increasing simulation duration. During these tests, simulation runtimes were recorded as model and watershed properties changed. The objective of this experiment was to understand the computational scaling of a loosely integrated architecture (i.e. the OpenMI) for semi-distributed rainfall/runoff applications. The HMS model implemented to provide a basis for comparison because it is a widely used, industry-standard hydrologic model. In this section we first introduce the study area, the Smith Branch Watershed near Columbia, South Carolina, and then describe the model from a mathematical and implementation perspective. Finally we describe the computational scaling tests.

3.1. Study Area

The study area for this work was the Smith Branch watershed located in Columbia, SC (Figure 3). The watershed has an area of 5.6 mi^2 and consists mostly of developed land (11% high intensity, 26% medium intensity, 37% low intensity, and 16% open space). The primary soil type in the watershed is sandy loam (70%), while 25% of the watershed is loam and 5% is sand. The watershed is gaged by a US Geological Survey monitoring station 02162093 "Smith Branch at North Main Street at Columbia, SC". This gauging station reports both water surface height and discharge every 15 minutes. We did not calibrate the models using this observed flow because the goal was to understand computational performance rather than simulate a particular event.

3.2. Model Description

A conceptual view of a rainfall/runoff system is presented in Figure 4. The models implemented in both HMS and OpenMI contain mathematical approximations for precipitation, infiltration, surface runoff, and streamflow. Precipitation was supplied as model input and obtained from a Next Generation Radar

(NEXRAD) precipitation product provided through the National Climatic Data Center (NCDC). Excess precipitation (P_e) is modeled using the Natural Resources Conservation Service (NRCS, formally SCS) Method for Abstraction (Soil Conservation Service, 1972; Chow et al., 1988). According to this method

$$P_e = P - I_a - F_a \tag{1}$$

where P is precipitation, I_a is initial abstraction, and F_a is continuing abstraction. F_a is estimated by a conceptual model

$$F_a = \frac{S(P - I_a)}{P - I_a + S} \tag{2}$$

where S is soil water storage. Furthermore, S is estimated based on an empirical equation

$$S = \frac{1000}{CN} - 10 \tag{3}$$

where CN is the curve number parameter which is a function of land cover and soil conditions (Soil Conservation Service, 1975). Lastly, I_a is estimated using an empirically derived relationship that states $I_a = 0.2S$.

These equations are solved to estimate the amount of excess precipitation (P_e) that will occur given a rainfall event. This method for estimating rainfall abstraction is commonly used in hydrologic engineering practice, which was the motivation for using it in our study. In addition, because the overall goal of the study is to test loose integration of hydrologic models model, predictive accuracy of the model itself was not our primary focus. Curve numbers where derived using geospatial analysis; spatial data layers of land cover and soil data were gathered from the Environmental Protection Agency (EPA) and NRCS, respectively, and curve number values were taken from the NRCS Technical Reference Manual 55: Urban Hydrology for Small Watersheds (Soil Conservation Service, 1975). Weight averaged curve numbers were calculated for each subbasin within the watershed using this data and approach.

The watershed response to a rainfall event was estimated using the NRCS Dimensionless Unit Hydrograph (Chow et al., 1988). This is a deterministic and lumped approach for estimating a streamflow hydrograph at the outlet of a subbasin given an excess precipitation hyetograph. To apply the unit hydrograph procedure, several parameters must first be derived: the time at which peak flow occurs (T_p), the peak flow rate (q_p), and the subbasin lag time (t_p). Alternatively, lag time can be represented as $0.6t_c$, where t_c is the time of concentration for the subbasin. Peak flow rate was estimated using

$$q_p = \frac{483.4A}{T_p} \tag{4}$$

where A is the watershed area in mi^2, T_p is in hours, and q_p has units of ft^3 s^{-1} in. T_p is defined as

$$T_p = \frac{t_r}{2} + t_p \tag{5}$$

where t_r is the rainfall duration in hours. An instantaneous streamflow hydrograph was derived for each subbasin using the unit ordinates of the NRCS dimensionless hydrograph and the calculated peak flow rate (q_p). Once formed, this instantaneous unit hydrograph was used to calculate a direct runoff hydrograph using Equation (6) where $P_{e,m}$ is the excess precipitation at time m and U_{n-m+1} is the unit hydrograph ordinate of the current index, n, minus $m + 1$ (Chow et al., 1988).

$$Q_n = \sum_{m=1}^{n<=M} P_{e,m} U_{n-m+1} \tag{6}$$

Flow is routed through the channel network using the Muskingum method. This method transforms the streamflow calculated at each subbasin outlet through a channel network to the watershed outlet. The Muskingum algorithm is derived from a variable discharge-storage relationship and considers the total water storage in a channel as a combination of wedge and prism volumes (Chow et al., 1988). The wedge shape accounts for the back water or flood wave effects, and is controlled by a weighting factor, X, where $0 \leq X \leq 0.5$. The prism storage represents the volume of water within a cross-section of the channel, and is weighted by the proportionality coefficient, K, that is approximated as the time it takes for a flood wave to travel through the reach (Mays, 2005). The total storage, S, is expressed as a combination of wedge and prism volumes

$$S_{j+1} - S_j = \frac{I_j + I_{j+1}}{2}\Delta t - \frac{Q_j + Q_{j+1}}{2}\Delta t \tag{7}$$

where I is inflow and Q is outflow at times j and $j+1$, in units of ft^3 s^{-1} . Given that storage can be expressed as a function of the K and X parameters, Equation (7) can be simplified to

$$Q_{(j+1)} = C_1 I_{(j+1)} + C_2 I_j + C_3 Q_j \tag{8}$$

where the C1, C2, and C3 coefficients are functions of K, X, and Δt (Chow et al., 1988).

3.3. Model Implementation

3.3.1. HMS Model

The process of creating the HMS model for Smith Branch is illustrated in Figure 5. It began with constructing hydrologic elements using the HMS Basin Model Manager. These elements represent specific parts of the overall hydrologic system. Linkages are used to connect elements in series and to define the order in which the system of equations is solved. To recreate the Smith Branch watershed, a set of subbasin, reach, and junction elements were connected in series. Once the watershed was assembled, process representations were chosen for each hydrologic element. As discussed in Section 3, infiltration was calculated using the NRCS Curve Number method, surface runoff was calculated using the NRCS Unit Hydrograph procedure, and channel routing was performed using the Muskingum method. Lastly, all input parameters were specified for each mathematical procedure. These input parameters were derived from geospatial data sets and were processed using Geographic Information System (GIS) software.

The HMS Time-Series Data manager was used to apply the measured rainfall values to the watershed in a spatially distributed manner. This was achieved by first creating a synthetic precipitation gage for each subbasin within the watershed. Next, incremental rainfall hyetographs were manually entered from the recorded NEXRAD precipitation data. The HMS Meteorologic Model manager was used to pair these rainfall hyetographs with their respective subbasins. Once complete, each subbasin within the watershed was associated with a unique rainfall hyetograph derived from radar data.

After all preparatory data procedures were complete, the HMS Control Specifications manager was configured to control model simulation. This tool enables users to specify simulation start and end date-times and also the simulation time interval. Lastly, a simulation run was created and executed. Simulation results were stored for each hydrologic element within the system, in the Hydrologic Engineering Center Data Storage System (HEC-DSS).

3.3.2. OpenMI Model

To construct the OpenMI model, components were first developed to perform infiltration, surface runoff, and channel flow calculations. Each process was defined by a specific mathematical relationship, described in the previous section: NRCS Curve Number method, NRCS Unit Hydrograph transformation, and Muskingum channel routing, respectively (Figure 6). A key aspect of the OpenMI is that components communicate with each other during model simulation on a time step basis. This communication is designated by the "request" and "reply" loop shown in the right column of Figure 6. This workflow is different than most traditional models that tend to execute each process for the entire simulation duration, sequentially. In contrast, once a simulation is initiated in the OpenMI model, the "downstream" component in the chain requests data from its "upstream" neighbor. In this case Channel Flow requests data from Surface Runoff. The Surface Runoff component is unable to supply values to Channel Flow because it to requires input from another component. Likewise, the Surface Runoff component will request data from Infiltration which subsequently requests data from Precipitation. Since Precipitation does not require input from any other component, it is able to respond to Infiltration with the requested data. The Infiltration component receives this data, processes it, and then sends the result to Surface Runoff. This is repeated until Channel Flow processes its result, at which time the entire sequence of events begins again for the next time step.

Components were developed using an abstract class based on the OpenMI Association Technical Committee's (OATC) IEngine interface called the Simple Model Wrapper (SMW) (Castronova and Goodall, 2010). The OATC IEngine was created to aid in the process of "wrapping" legacy models into OpenMI-compliant components. In previous work we showed that the IEngine interface can be simplified for creating new, process-level model components using the concept of an abstract class. Therefore, the SMW was developed to enable model developers to quickly create new process-level components by simplifying the IEngine interface to an abstract class where the model developer overrides three methods: Initialize, PerformTimeStep, and Finish. Furthermore, model attributes are specified in an XML configuration file. The Initialize method is used to setup a model component prior to model simulation. It can be used to read input data, parameterize the model, establish initial conditions, or perform similar setup operations. The PerformTimeStep method is used to execute calculations that occur during simulation runtime. This method starts by requesting input values from "upstream" components, then performing a computation using these values, and finally supplying the results to "downstream" components. The Finish method is called after a simulation ends and can be used to write model output files and release allocated memory.

Rainfall data was supplied to the model configuration using a component designed to read time-series data stored in Water Markup Language (WaterML) format; an XML-based standard for storing hydrologic time-series values (Valentine and Zaslavsky, 2009). Prior to simulation, this component creates exchange items by reading a directory of WaterML files. It uses the information within them to establish input and output exchange items, element sets, and time horizons. Values are stored in a buffer (i.e. Oatc.SmartBuffer) so they can be retrieved for specific times, upon request. In the case that a component requests values for a time that does not match those included in the input files, this component performs a linear time interpolation from the known values, to respond to the request.

The amount of excess precipitation was determined using the NRCS Curve Number method. During the initialization phase, curve number values for each subbasin are read from an input shapefile and stored in memory so they can be accessed during simulation. Next, input and output exchange items are defined by reading information contained within the input shapefile. Lastly, potential infiltration and initial abstraction are calculated for each subbasin using Equations (1) through (3). During model simulation, the PerformTimeStep method requests values from the rainfall component. It then calculates how much excess precipitation will occur, based on these values and subbasin properties. The component can then supply excess precipitation for each subbasin in the watershed to the NRCS Unit Hydrograph component. Once model simulation is complete, the Finish method is called to write results to an output file.

Excess precipitation was transformed into subbasin outflow using the NRCS Unit Hydrograph method. During the Initialize phase, a unique unit hydrograph is created for each subbasin in the watershed based on their unique physical properties. This is done by first extracting parameters from the input shapefile, then calculating the time in which peak flow occurs using Equation (5), and finally calculating the peak flow rate for this time using Equation (4). Once these values have been calculated, a unit hydrograph is established using the dimensionless unit ordinates provided by the National Engineering Handbook (Soil Conservation Service, 1972). These unit hydrographs are utilized in PerformTimeStep method to calculate the outflow at the current time. This is done using Equation (6), where $P_{e,m}$ is the excess rainfall that is provided by the NRCS Curve Number component. In the Finish method, subbasin outflows are written to disk and internal component objects are released from memory.

Lastly, the subbasin outflows were routed through the stream network using the Muskingum Routing method. During the component's Initialize phase, channel parameters such as storage and proportionality coefficients, X and K, are read from an input shapefile. A network is also created from the input shapefile by associating each reach with upstream and downstream neighbors. During the PerformTimeStep phase, this component retrieves outflow from each subbasin and routes them through the stream network using Equation (8). The Finish method is used to save the routed streamflow to a specified output file.

3.4. Performance Tests

Performance tests were conducted to evaluate the potential computational costs introduced by the OpenMI communication paradigm when simulating a semi-distributed rainfall/runoff system. In the first test, the number of modeling units per simulation was varied. By increasing the number of modeling units, we were able to evaluate how each model operates under different data loading conditions. For the OpenMI model, this increase in modeling units is not only more computationally demanding, it also results in larger sized (in terms of memory) data transfers between model components. We termed this test the "load test" because it increases the size of data that components must pass between one another during a simulation run (Meier et al., 2007). The second test was designed to evaluate how each model responds to longer simulation durations. To do this, we varied the number of time steps performed in a simulation from 100 to 10,000, while maintaining a constant time step interval. Since a constant time step was maintained, synthetic rainfall data was required as input for both models to extend simulation duration. We termed this test the "endurance test" because it evaluates the ability for models to handle a large number of data transfers in a single simulation (Meier et al., 2007).

Tools were used, such as Arc Hydro (Maidment, 2002) and ArcGIS®, to subdivide the watershed into 1, 50, 100, 150, and 200 subbasins. It was not practical to subdivide the Smith Branch watershed into more that 200 sub-basins, but we did extend the load test to a larger number of model elements into the thousands by considering a synthetic watershed. Several scripts were written to automate the necessary geoprocessing steps required to create model inputs. This included the creation of input shapefiles with model parameters such as CN and T_c for both the HMS and OpenMI. The OpenMI components utilized these shapefiles as model input. In contrast, HMS model simulations necessitated supplementary scripts to extract the element set data from the various shapefiles, and programmatically write model inputs. The end result were identical model compositions within each software environment. The model runtimes used in our analysis were taken from reported values within HMS and OpenMI simulation output files.

4. Results and Discussion

The first objective of our study was to ensure that both models, the one built with HMS and the other built with OpenMI, produced the same output predictions for the watershed system. Figure 7 shows that both models calculated identical values for cumulative excess precipitation, surface runoff, and streamflow. This confirms that no errors were introduced as a result of decoupling the physical system into individual OpenMI-compliant computational components. The second objective was to quantify the overhead introduced during model simulation by the OpenMI communication standard. This objective is subdivided into two distinct tests (1) the load test and (2) the endurance test.

Results from the load test where the Smith Branch watershed was delineated into a varying number of subbasins revealed that, as the number of computational elements increased, the OpenMI's performance remained comparable to that of the HMS (Figure 8). In fact, the OpenMI model completed its simulation runs faster than the HMS for almost every scenario. The shaded regions around each line define the range of runtimes for each test case. This variability in runtime was caused primarily from the limited precision available for measuring runtimes, in particular for the HMS case, but also variability in computing resources available during each model run. The results suggest that the HMS runtime is following a linear trend for 100, 150 and 200 subbasins where the model completed in one second for both the 1 and 50 subbasin cases. The OpenMI shows some evidence of nonlinearity as the number of subbasins increases, and we present a separate test later in this section for extending the study beyond 200 model elements to further explore this finding. Analysis of just the test results and not extrapolating beyond 200 subbasins, however, suggests that the overhead introduced by component-to-component data transfers in the OpenMI are minimal. The small range of variability for the OpenMI simulations is a result of our ability to precisely measure execution times for the different simulations. In contrast, the HMS's variability is much larger, due primarily to the lack of precision in which execution times could be recorded. This lack of precision also offers an explanation for the seemingly inconsistent behavior of the HMS. Since the actual recorded execution times for the HMS model vary irregularly, we looked at how the shaded region trends as loading increases.

Figure 9 illustrates memory consumption for each system during the loading test. Values were determined by manually examining CPU resources while simultaneously opening, loading, and executing simulations. The "Startup" and "Model Load" phases of simulation represent the opening of the application itself and loading of the various models, respectively. Their values remain fairly constant for both systems, even as the number of hydrologic units increases. In contrast, the "Simulation" portion of model execution changed in each scenario. Both models follow a linear and fairly constant trend with a small slope during this phase. This suggests that as the number of data points increases beyond 200 elements, we can expect both models to behave in a similar way.

Because the loading test suggested that a nonlinear trend may occur in the OpenMI system as the number of modeling units per simulation continues to increase, we extended the load test to include a larger number of model elements. Synthetic element sets were created so that the number of modeling units could extend beyond the range typically used for semi-distributed modeling. This large-scale loading test was only done for the OpenMI model, as it was not possible to for us to extend the HMS into the range of thousands of subbasin units. Figure 10 illustrates how the OpenMI model execution time varies as the number of modeling elements increases beyond the range of the previous test. The first plot (left) shows the total model simulation time compared to the component-to-component communication time, which was estimated by subtracting the sum of the time spent within each component's *Initialize*, *PerformTimeStep*, and *Finish* methods from the total simulation time. It is clear from this analysis that the component-to-component communication represents only a small fraction of the total model execution time. Also, we did not see any evidence of nonlinearity in the OpenMI communication itself so that any nonlinearity present in Figure 8 is attributed to the scaling of the individual model components and not the OpenMI communication paradigm.

The second plot in Figure 10 (right) provides further insight into computational performance by presenting the simulation runtime for each component, which equals the sum of the time spent within that component's *Initialize*, *PerformTimeStep*, and *Finish* methods. This analysis shows that the majority of the model simulation time is due to the Muskingum component and, more specifically, to this component's *PerformTimeStep* method. This finding illustrates how the computational expense of a single component can have a profound impact on the overall execution speed of the model. One advantage of componentizing the model is that it is relatively easy to isolate computational bottlenecks within the system by tracking the runtime of each component within the system, and because each component is loosely coupled within the system, efforts can be made to address computational inefficiencies of a single component in isolation of the larger system.

In the endurance test, the Smith Branch watershed was delineated into 15 subbasins and simulations were executed over varying time durations, all having the same time step. The results show that the OpenMI model scales well in comparison to the HMS as simulation duration increases (Figure 11). It was predicted that the runtime of the OpenMI model would drastically increase as the number of time steps per simulation increased. This assumption was made on the basis that, if a performance lag exists during component communication, then this lag will become more pronounced as the number of communications increases. In fact, the OpenMI model completed in approximately half the time as HMS for a simulation that required 10,000 time steps. As previously stated, the shaded region of the figure represents variability in the simulation runtimes that resulted primarily from the limited precision in which runtimes could be measured and, to a lesser extent, variability in computing resources available when an individual model run was completed. The upper and lower bounds of these regions are defined by the minimum and maximum recoded execution times of the respective modeling systems. The HMS scaling follows an irregular trend that appears to be nonlinear with the exception of the result for 5,000 time steps. In contrast, the OpenMI scaling appears to be linear over the entire test range. This result was expected because the same calculation is evaluated on each time step of the model run, therefore 100 more time steps would require 100 more calculations per component. The variability in the OpenMI model's execution times are very small because it was possible to record simulation runtimes with high accuracy, whereas the HMS variability is much larger because our method for measuring simulation runtimes of the HMS model lacked precision. Considering the lower limit of these regions, both models appear to follow a linear trend, although the HMS runtime increases more rapidly than that of the OpenMI. From this result we can infer that as the simulation length increases, the OpenMI model will continue to scale better that the HMS.

Figure 12 illustrates the allocated memory for each model as three distinct phases of simulation: "Startup," "Model Load," and "Simulation." Both models respond similarly under the "Startup" and "Model Load" phases of the test. These trends are expected to continue linearly as the number of time steps exceeds 10,000. The primary concern was how the OpenMI architecture stores data over long periods of time. As expected, each model consumes more memory during the "Simulation" phase. Both the HMS and OpenMI models, however, require additional resources at a similar rate. The outcome of this test suggests that performance lags do not result from data storage over a long simulation duration.

When interpreting the results from the semi-distributed load and endurance tests we must consider that HMS is a fully implemented model, meaning it includes additional data logging and post processing routines. The OpenMI implementation was created for this study and is a very light-weight implementation that does not include robust error handling, data logging, or user customization. These differences may influence the scalability of each model. For this reason, the results should not be interpreted to mean that OpenMI is faster than HMS, because the two models do not include the exact same functionality. When designing these experiments, the goal was to quantify how the two models scaled in a general sense as we increased data communication loads and frequencies. The most important discovery, therefore, is that the OpenMI model scaled in a similar way to the HMS. Furthermore, the OpenMI architecture does not introduce any significant computational overhead.

We did not anticipate that both versions of the model would scale similarly, instead we expected that the additional communication introduced by the loosely integrated approach would result in performance loss. This performance loss, we thought, would stem from inefficiencies in how data is communicated between OpenMI model components during runtime. The load and endurance tests that were performed indicate that the mechanism by which the OpenMI shares data amongst components introduces insignificant lag for semi-distributed applications. However, because it is possible that the data communicated between components was sufficiently small that lags appeared to be non-existent, we performed another test to evaluate the OpenMI under larger loading conditions. This second test showed that the OpenMI communication had a small impact on the overall model runtime and scaled with a linear trend.

5. Summary, Conclusions, and Future Work

The primary motivation for this study was to improve understanding of the computational efficiency of a loose integration paradigm, specifically the OpenMI version 1.4, for modeling hydrologic systems. Our goal was to first show that it is possible to construct a process-level loosely integrated hydrologic model, and second to quantify the computational cost associated with message passing. To do this, a semi-distributed rainfall/runoff watershed model was developed as a set of OpenMI components and evaluated in two performance tests. For comparison purposes, we constructed a similar representation using an industry-standard watershed model: HMS. In the first test, the size of messages passed between components was gradually increased. This test addressed the OpenMI's ability to manage the transfer of various sized data packets between components. The OpenMI model itself was then tested under larger loading conditions, outside the practical range for semi-distributed watershed modeling applications, to further investigate its scalability. In the second test, the number of messages passed between model components was increased to test its impact on simulation runtime. This was accomplished by extending the simulation duration while maintaining a constant time step. This test addressed the efficiency of the OpenMI's request and reply mechanism. Together, these experiments allowed us to quantify how the OpenMI model scaled as the size and number of messages passed between model components increased.

We found that the OpenMI-based model was able to produce identical values to the HMS model, a result that we expected and that was necessary to show prior to performance testing. The load test evaluated how efficiently data is shared among components and quantified the effect that the transfer of increasingly sized data packets has on model performance. We found that the OpenMI model runtimes scaled well in comparison to the HMS. However, the shaded regions in Figure 8 suggested that the OpenMI may follow an nonlinear trend as the number of modeling units is extended beyond what was used in this study. We were able to further explore this finding by extending the loading conditions beyond the range of semi-distributed applications using synthetic modeling units. We found from this experiment that the nonlinear trend of the OpenMI model was caused by the computational expense of the Muskingum component and not the OpenMI communication mechanism. By decomposing the system into components with standard interfaces, it is trivial to track the computational time spent in each component's key methods during the simulation run. This can provide valuable information for addressing model bottlenecks and improving overall model performance. The endurance test evaluated how the OpenMI performed for increasing time durations. The results of this test (Figure 11) again show that the OpenMI scales favorably in comparison to the HMS. In fact, it appears from our tests to follow a linear trend, suggesting that as simulation duration continues to increase, runtime will still be within an acceptable range. Overall, the performance experiments of adjusting the size and quantity of messages passed within an OpenMI configuration showed no significant performance issues within ranges typical for semi-distributed watershed modeling. Future work should extend these tests to ranges more common for more computationally demanding hydrologic modeling paradigms.

Acknowledgments

This work was supported by the National Science Foundation under the grant NSF EAR 0622374 “GeoInformatics: Consortium of Universities for the Advancement of Hydrologic Sciences, Inc. (CUAHSI) Hydrologic Information Systems.”

References

Ahrends, H., Mast, M., Rodgers, C., Kunstmann, H., 2008. Coupled hydrologicaleconomic modelling for optimised irrigated cultivation in a semi-arid catchment of west africa. Environmental Modelling & Software 23 (4), 385–395.

Al-Abed, N., Abdulla, F., Khyarah, A. A., 2005. GIS-hydrological models for managing water resources in the zarqa river basin. Environmental Geology 47 (3), 405–411.

Anderson, M. L., Chen, Z., Kavvas, M. L., Feldman, A., 2002. Coupling HEC-HMS with atmospheric models for prediction of watershed runoff. Journal of Hydrologic Engineering 7 (4), 312.

Argent, R. M., Voinov, A., Maxwell, T., Cuddy, S. M., Rahman, J. M., Seaton, S., Vertessy, R. A., Braddock, R. D., 2006. Comparing modelling frameworks - a workshop approach. Environmental Modelling & Software 21 (7), 895–910.

Barthel, R., Janisch, S., Schwarz, N., Trifkovic, A., Nickel, D., Schulz, C., Mauser, W., 2008. An integrated modelling framework for simulating regional-scale actor responses to global change in the water domain. Environmental Modelling & Software 23 (9), 1095–1121.

Brinkman, R., Gregersen, J. B., Hummel, S., Westen, S. J., 2005. Part c - the org.OpenMI.Standard interface specification. The OpenMI Document Series, 1–79.

Castronova, A. M., Goodall, J. L., July 2010. A generic approach for developing process-level hydrologic modeling components. Environmental Modelling & Software 25 (7), 819–825.

Chow, V. T., Maidment, D. R., Mays, L. W., 1988. Applied Hydrology. McGraw-Hill Inc, Boston, Massachusetts.

Chu, X. F., Steinman, A., Feb. 2009. Event and continuous hydrologic modeling with HEC-HMS. Journal of Irrigation and Drainage Engineering 135 (1), 119–124.

Feldman, A. D., 2000. Hydrologic modeling system HEC-HMS technical reference manual. Technical Reference Manual CPD-74B, U.S. Army Corps of Engineers, Hydrologic Engineering Center, Davis, CA, http://www.hec.usace.army.mil/software/hec-hms/documentation/CPD-74B_2000Mar.pdf.

Govindaraju, M., Katz, D. S., Kohl, J. A., Krishnan, M., Kumfert, G., Larson, J. W., Lefantzi, S., Lewis, M. J., Malony, A. D., McInnes, L. C., Allan, B. A., Nieplocha, J., Norris, B., Parker, S. G., Ray, J., Shende, S., Windus, T. L., Zhou, S. J., Armstrong, R., Bernholdt, D. E., Bertrand, F., Chiu, K., Dahlgren, T. L., Damevski, K., Elwasif, W. R., Epperly, T. G. W., 2006. A component architecture for high-performance scientific computing. International Journal of High Performance Computing Applications 20 (2), 163–202.

Gregersen, J. B., Gijsbers, P. J. A., Westen, S. J. P., 2007. OpenMI: open modelling interface. Journal of Hydroinformatics 9 (3), 175–191.

Holzworth, D. P., Huth, N. I., de Voil, P. G., 2010. Simplifying environmental model reuse. Environmental Modelling & Software 25 (2), 269–275.

Kennen, J. G., Kauffman, L. J., Ayers, M. A., Wolock, D. M., Colarullo, S. J., 2008. Use of an integrated flow model to estimate ecologically relevant hydrologic characteristics at stream biomonitoring sites. Ecological Modelling 211 (1-2), 57–76.

Knebl, M., Yang, Z., Hutchison, K., Maidment, D., 2005. Regional scale flood modeling using NEXRAD rainfall, GIS, and HEC-HMS/RAS: a case study for the san antonio river basin summer 2002 storm event. Journal of Environmental Management 75 (4), 325–336.

Kunstmann, H., Jung, G., Wagner, S., Clottey, H., 2008. Integration of atmospheric sciences and hydrology for the development of decision support systems in sustainable water management. Physics and Chemistry of the Earth 33 (1-2), 165–174.

Löwy, J., 2005. Programming .NET components, 2nd Edition. O'Reilly Media Inc.

Maidment, D., 2002. Arc hydro : GIS for water resources. ESRI Press, Redlands Calif.

Maxwell, R., Chow, F., Kollet, S., 2007. The groundwater land-surface atmosphere connection: Soil moisture effects on the atmospheric boundary layer in fully-coupled simulations. Advances in Water Resources 30 (12), 2447–2466.

Mays, L. W., 2005. Water Resources Engineering, 2005th Edition. John Wiley and Sons, Inc, Hoboken.

Meier, J., Farre, C., Bansode, P., Barber, S., Rea, D., Sep. 2007. Microsoft patterns and practices. http://msdn.microsoft.com/en-us/library/bb924359.aspx.

Mölders, N., Rühaak, W., 2002. On the impact of explicitly predicted runoff on the simulated atmospheric response to small-scale land-use changesan integrated modeling approach. Atmospheric Research 63 (1-2), 3–38.

Moore, R., Gijsbers, P., Fortune, D., Gregersen, J., Blind, M., 2005. The OpenMI document series: Part A - scope for the OpenMI (version 1.0). HarmonIT.

Morton, Y., Troy, D., Pizza, G., 2003. A state-based modelling approach to develop component-based control software for flexible manufacturing systems. International Journal of Computer Integrated Manufacturing 16 (4-5), 292–306.

Scharffenberg, W., Fleming, M., 2009. Hydrologic modeling system HEC-HMS user's manual. Technical Reference Manual CPD-74A, U.S. Army Corps of Engineers, Hydrologic Engineering Center, Davis, CA.

Singh, V. P., Woolhiser, D. A., Aug. 2002. Mathematical modeling of watershed hydrology. Journal of Hydrologic Engineering 7 (4), 270–292.

Soil Conservation Service, 1972. Hydrology. In: National Engineering Handbook. section 4. U.S. Department of Agriculture, Washington, D.C.

Soil Conservation Service, 1975. Urban hydrology for small watersheds. Technical Reference 55, U.S. Department of Argiculture, Washington, D.C.

Sui, D. Z., Maggio, R. C., 1999. Integrating GIS with hydrological modeling: practices, problems, and prospects. Computers, Environment and Urban Systems 23 (1), 33–51.

Valentine, D., Zaslavsky, I., Jun. 2009. Introduction to WaterML schema. In: CUAHSI WaterML 1.0 Specification. Part 1. CUAHSI universities allied for water research.

Yu, Z., Pollard, D., Cheng, L., 2006. On continental scale hydrologic simulations with a coupled hydrologic model. Journal of Hydrology 331 (1-2), 110–124.

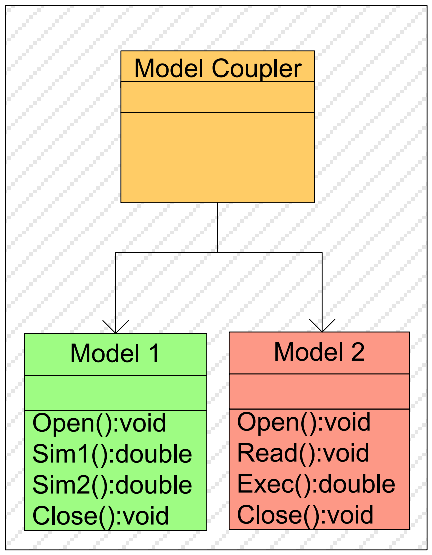

Figure 1: In a tightly integrated system, the Model Coupler class directly points to the methods defined within each Model.

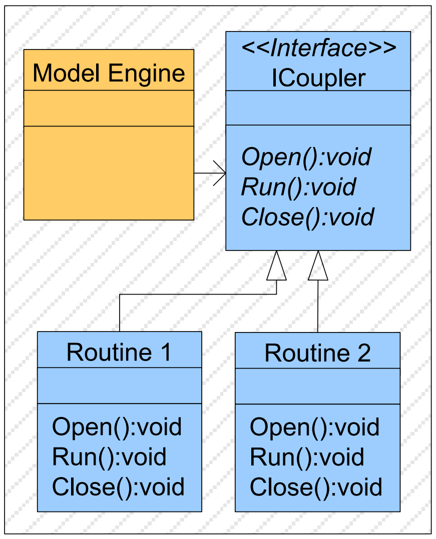

Figure 2: In a loosely integrated system, the ICoupler interface defines the methods required by the Model Engine, but they are implemented by external Routines.

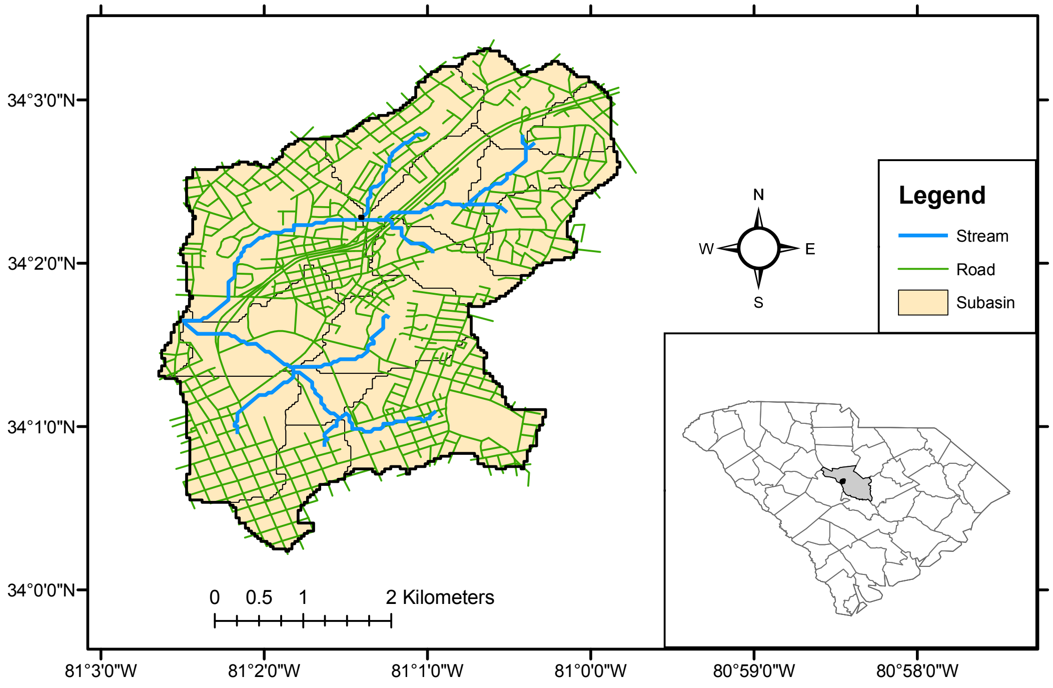

Figure 3: Schematic view of the Smith Branch watershed, including the derived subbasins and reach network.

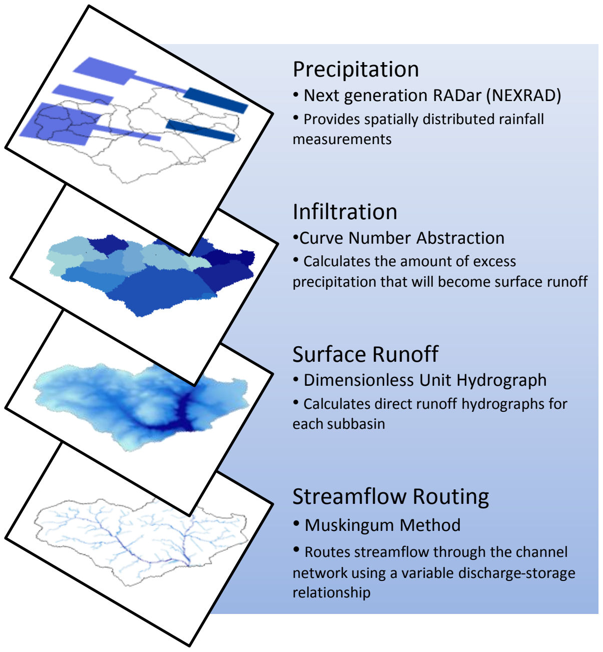
Precipitation
• Next generation RADar (NEXRAD)
• Provides spatially distributed rainfall measurements
Infiltration
•Curve Number Abstraction
• Calculates the amount of excess precipitation that will become surface runoff
Surface Runoff
• Dimensionless Unit Hydrograph
• Calculates direct runoff hydrographs for each subbasin
Streamflow Routing
• Muskingum Method
• Routes streamflow through the channel network using a variable discharge-storage relationship

Figure 4: A conceptual representation of a simple rainfall/runoff system

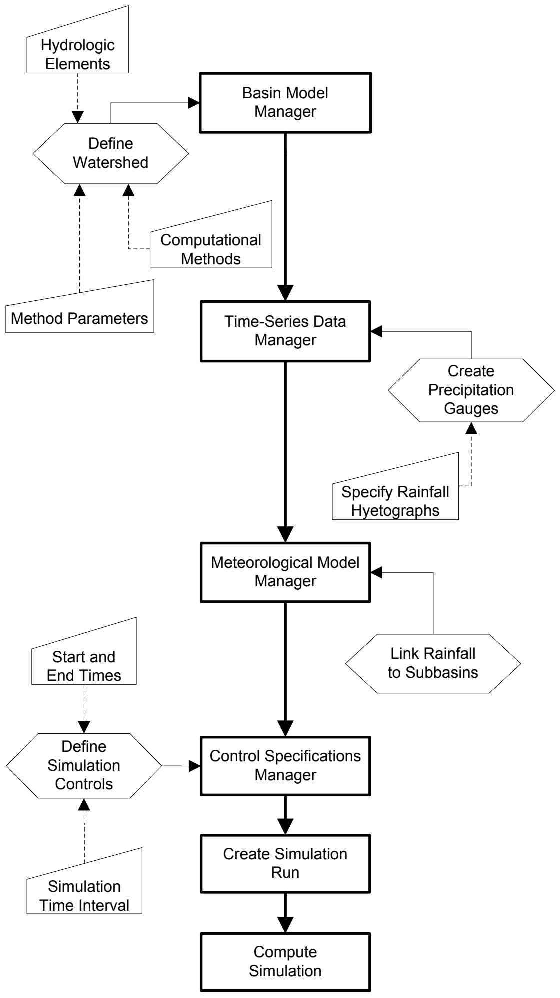

Figure 5: Flow chart showing the model development procedure within the Hydrologic Modeling System.

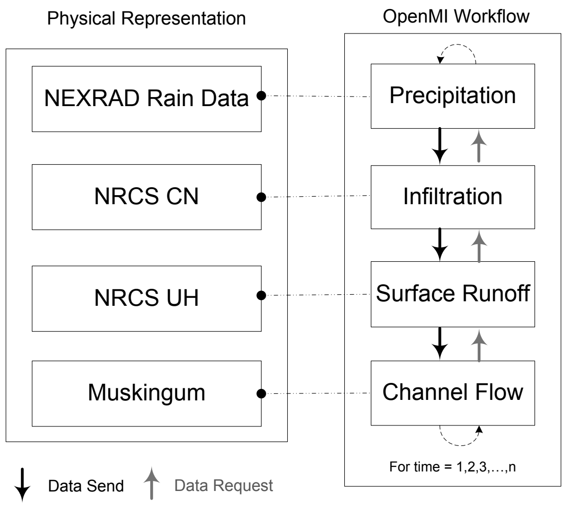

Figure 6: The work flow of the loosely integrated model, showing the interaction of each component with a single physical process representation.

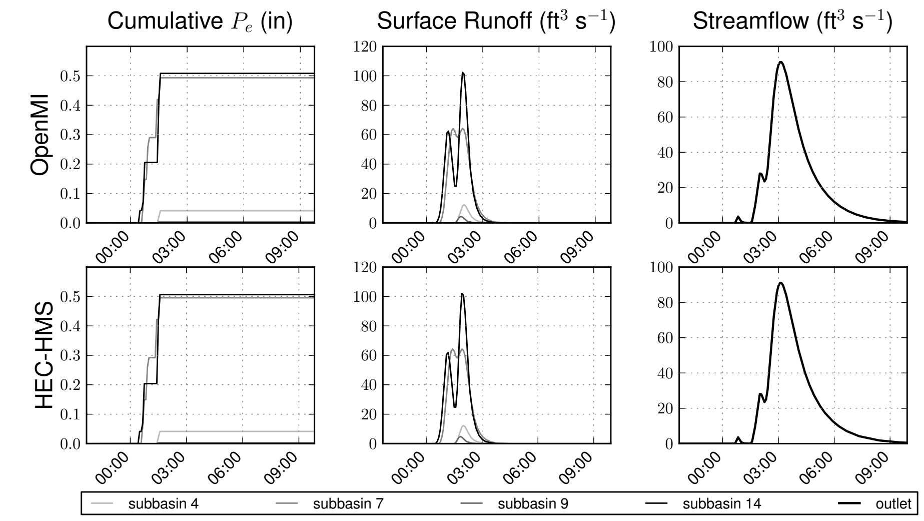

Figure 7: The results produced by each modeling system. The OpenMI values are given in the first row and HMS values are in the second. Cumulative P_e and surface runoff are shown for the contributing subbasins, and streamflow is at the outlet of the watershed.

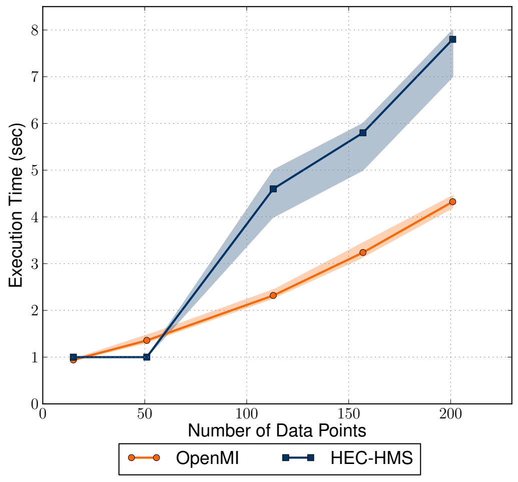

Figure 8: The load test shows how model run time is effected as the number of computational elements increases. The shaded regions indicate variability in simulation run time, bounded by the minimum to maximum recorded values.

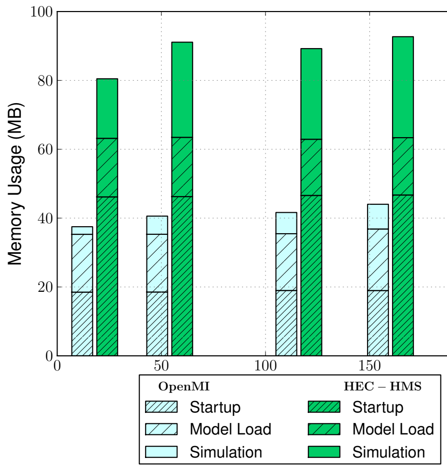

Figure 9: The memory allocated during each phase of model simulation, under various loading conditions.

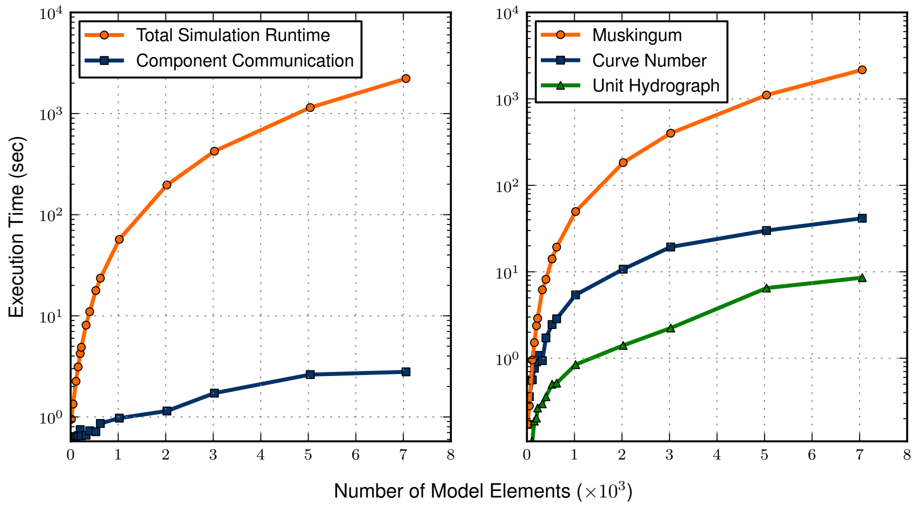

Figure 10: The large-scale loading test illustrates how the OpenMI model scales when the number of model elements extends into the thousands. The plot on the left gives the total simulation time for the composition and the fraction of time spent outside of the component's *Initialize*, *PerformTimeStep*, and *Finish* methods and therefore attributed to component-to-component communication. The plot of the right presents the total time spent within each component's *Initialize*, *PerformTimeStep*, and *Finish* methods.

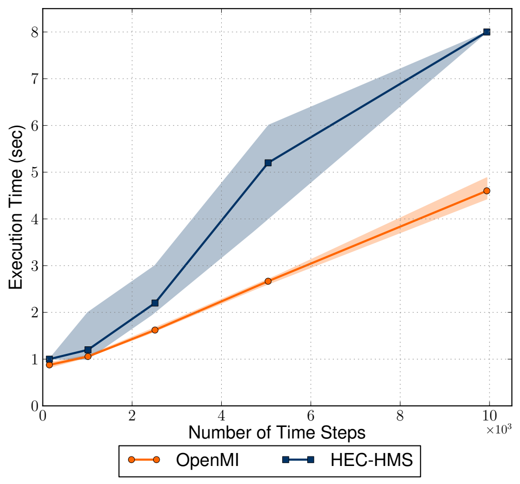

Figure 11: The endurance test shows how model run time is effected by an increasing number of simulation time steps. The shaded regions indicate variability in simulation run time, bounded by the minimum and maximum recorded values.

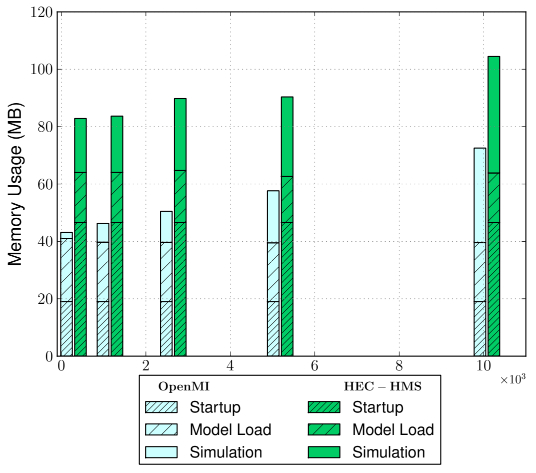

Figure 12: The memory allocated during each phase of model execution, for various simulation durations.