

California State University, San Bernardino

CSUSB ScholarWorks

Theses Digitization Project

John M. Pfau Library

2003

Simulation of soil water movement model (SWaMM) using the Spider Distributed System

Li Wang

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Wang, Li, "Simulation of soil water movement model (SWaMM) using the Spider Distributed System" (2003). *Theses Digitization Project*. 2419.

<https://scholarworks.lib.csusb.edu/etd-project/2419>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

SIMULATION OF SOIL WATER MOVEMENT MODEL (SWaMM)
USING THE SPIDER DISTRIBUTED SYSTEM

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science


by
Li Wang
June 2003

SIMULATION OF SOIL WATER MOVEMENT MODEL (SWaMM)
USING THE SPIDER DISTRIBUTED SYSTEM

A Project
Presented to the
Faculty of
California State University,
San Bernardino


by
Li Wang
June 2003

Approved by:


Dr. Arturo I. Concepcion, Chair, Computer Science

Date

04 Jun 2003


Dr. George M. Georgiou


Dr. Ernesto Gomez

ABSTRACT

Physical modeling is a technique of reproducing the behavior of a phenomenon on a more convenient geometrical scale. The prediction and understanding of soil water movement by the mathematical modeling process play an important role in agriculture, hydrology and environment science for designing and testing various management schemes on irrigation or controlling chemicals environmental hazards. This project simulates the Soil Water Movement Model on Spider II distributed system and the model comes from the Department of Environment Sciences, UCR. The Soil Water Movement Model, which consists of a partial differential equation and several auxiliary conditions, is solved by the alternating-directions implicit (ADI) difference method. Based on the sequential algorithm, this project develops a parallel and distributed algorithm for Soil Water Movement Model on Spider II system, which is a distributed computing research project ongoing in the Department of Computer Science, California State University San Bernardino. This project implements the parallel and distributed algorithm using multi-thread networking programming skill, which eliminated

the interprocess communication between the parent and child when we use fork to create new processes. Several testing grid sizes of soil profile were used and all results of computation time, speedup and efficiency show that Spider II system has better overall performance over the sequential program. Finally, a Java-based user interfaces and execution views are implemented as well to display the dynamic 2D diagram for Soil Water Movement Model.

The support of the National Science Foundation under the award 9810708 is gratefully acknowledged.

ACKNOWLEDGMENTS

The completion of this project reflects the assistance and advice of a number of people who have markedly improved the final product. I would like to express my sincere gratitude to my mentor, Dr. Arturo I. Concepcion, for his invaluable assistance in the initial design of the project coupled with suggestions and encouragements during the programming and writing phases of this project. I appreciate his allowing me to do the research in my own way, giving me the freedom and opportunity to develop my own thoughts and ideas.

I would also like to thank Dr. Laosheng Wu, an associate professor at Department of Environment Science in UCR for his interesting in my work. I thank him for his help and for our many discussions concerning soil science.

The people that have helped me in my research and enriched my life the most are Xiao Zhang, Xiwei Wu, Jianhua Ruan, Zhuo Chen, Yan Jiang and Hao Jia. Without their friendship and great help I would not have come to this point in my life.

Special thanks goes to my husband, Dr. Guanglong Feng, who has encouraged me through my schooling. Thank you for your never-ending faith, support and love. Thanks for inspiring me to choose my career again.

Finally, thanks to my farther and mother - in law for enduring through one year of hard work to take care of my baby, and my parents who have been very supportive of my goals in life. I own all my triumphs and smarts to you and the example you set for me.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGMENTS	v
LIST OF TABLES	x
LIST OF FIGURES	xi
CHAPTER ONE: INTRODUCTION	
1.1 Introduction	1
1.2 Purpose of the Project	3
1.3 Organization of the Documentation	5
CHAPTER TWO: SOFTWARE REQUIREMENTS SPECIFICATION	
2.1 Introduction	6
2.1.1 Scope	6
2.1.2 Overview	7
2.2 Overall Description	8
2.2.1 Introduction to Model	9
2.2.2 Product Perspective	14
2.2.3 Product Functions	19
2.2.4 User Characteristics	21
2.2.5 Constraints	22
2.3 Specific Requirements for External Interface	22
CHAPTER THREE: SOFTWARE DESIGN	
3.1 Distributed Algorithm	31

3.2 Software Archeticture	41
3.2.1 Classes <i>WaterMovement</i> and <i>WaterMovement_slv</i>	41
3.2.2 Classes <i>OSB</i> and <i>OSB_slv</i>	45
3.2.3 Classes <i>TriMat_Constant</i> and <i>EquationGroup</i>	45
3.3 Detailed Design	47
3.3.1 Thread <i>Sender</i>	47
3.3.2 Thread <i>Getter</i>	49
3.3.3 Thread <i>Receiver</i>	50
3.3.4 Thread <i>Calculator</i>	53
3.3.5 Thread <i>Returner</i>	53
CHAPTER FOUR: SYSTEM TESTING AND PERFORMANCE ANALYSIS	
4.1 Testing Environments	55
4.2 Testing Methods	56
4.3 Testing Results	58
4.3.1 Computation Time	58
4.3.2 Speedup and Efficiency	60
CHAPTER FIVE: MAINTENANCE MANUAL	
5.1 System Requirements	65
5.2 Obtaining a Copy	65
5.3 Directory Organization	66
5.4 Checking for Thread Support	67
5.5 Installation of Spider II System	69

5.6 Running Soil Water Movement Model	72
5.7 Extention of Soil Water Movement Model	74
5.7.1 Client Program	74
5.7.2 Server Program	75
CHAPTER SIX: CONCLUSIONS AND FUTURE DIRECTIONS	
6.1 Conclusions	78
6.2 Future Directions	81
APPENDIX A: GLOSSARY	83
BIBLIOGRAPHY	90

LIST OF TABLES

Table 1. Comparison of Execution Time	59
Table 2. Comparison of Performance	61

LIST OF FIGURES

Figure 1.	The Architecture of Spider II	1
Figure 2.	Two-Dimensional Water Movement Model ...	10
Figure 3.	The Finite Difference Grid for Two-Dimensional Space	11
Figure 4.	The Distributed Algorithm	15
Figure 5.	Deployment Diagram	17
Figure 6.	User Case Diagram	19
Figure 7.	Welcome Web Page	24
Figure 8.	Introduction Web Page for Model	25
Figure 9.	Web Page to Input the Initial Condition	26
Figure 10.	Web Page for Monitoring and Displaying	29
Figure 11.	Work Cycle of Threads	39
Figure 12.	Component Architecture of Software	43
Figure 13.	Tridiagonal Matrix	44
Figure 14.	Pseudo Code of Thread <i>Sender</i>	49
Figure 15.	The part of Pseudo Code of Thread <i>Getter</i>	50
Figure 16.	Pseudo Code of Thread <i>Receiver</i>	51
Figure 17.	Pseudo Code of Thread <i>Calculator</i>	52
Figure 18.	Pseudo Code of Thread <i>Returner</i>	54
Figure 19.	Comparison of Execution Time	60

Figure 20. Comparison of Speedup	62
Figure 21. Comparison of Efficiency	63
Figure 22. Program <i>thread1.c</i>	67
Figure 23. Test Result for Program <i>thread1.c</i>	68
Figure 24. Graph of Soil Water Movement Model	73

CHAPTER ONE

INTRODUCTION

1.1 Introduction

Spider is an object-oriented distributed computing research project in the Department of Computer Science, California State University San Bernardino (CSUSB). It was initially proposed by Han-Sheng Yuh in his Master's thesis in 1997 [2]. And thereafter it has been further improved and implemented by Koping Wang as a Spider II distributed computing system in his Master's Project [3]. Spider II is developed in C++ and runs on Unix or Unix-like platform. The work is underway for SpiderIII, which is an agent-based distributed computing system [4].

Spider II system consists of four major components:

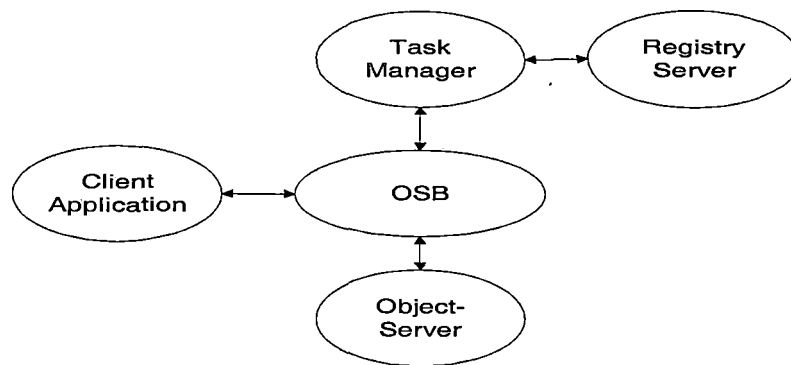


Figure 1. The Architecture of Spider II

Task Manager, Registry Server, Object Server Broker, and Object-Servers.

Task Manager: In order to manage the available servers for each task, Task Manager keeps track of all task activities.

Registry Server: The function of the Registry Server is to provide a list of available Object-Servers to the Task Manager. Registry Server auto-detects Object-Servers whether they are up or down and checks the percentage of utilization of the CPU usage. After detection, Registry will have a list of available Object-Servers' information. This information is sent to the Task Manager.

Object Server Broker: Object Server Broker plays an important role in the Spider II system. It submits the job to the Task Manager for requesting the available Object-Servers, activates the remote Object-Servers, distributes data to Object-Servers and collects computing results from Object-Servers and notifies Task Manager that the job is finished.

Object-Server: Object-Servers are the workstations on a network actually doing the computation.

Mirrors for both Task Manager and Registry Server were introduced in the Spider II system. A failure of the task

manager or registry server will activate its Mirror and all further communication is then redirected to the mirror.

In order to test the performance of Spider II system, three distributed applications are developed in Wang's project [3]: distributed matrix multiplication, distributed prime number search, and distributed Quick Sort. The first two applications show Spider II running on a group of low-end workstations on a network has better performance than a single high-end servers. For the third application, Spider II is slower but the increase in computational time for sorting very large number of data items is less than in a single high-end server.

However, those three simply applications are not enough to prove the computing applicability of Spider II.

1.2 Purpose of the Project

This project implements a real application on the Spider II, which is a simulation of Soil Water Movement Model.

More than 80% of total available water resource is being used in agriculture to meet huge food requirements of expanded population. Water shortage is becoming much more serious all over the world. Optimal water management is

very vital for higher yields and water use efficiency and to minimize water quality degradation.

Since prediction and understanding the dynamics of soil water movement processes can guide farmers and policy makers in making decisions that optimize the dual goal of high crop yield, high water use efficiency and low environmental degradation, a simulation model is developed to quantitatively describe and predict soil water spatial and temporal movement after an irrigation. There are many scientists in agriculture, hydrology, and environmental sciences who are working on calculating the model [6]. The soil water movement model implemented this project is come from Department of Environments, UCR.

Although the numerical model has been universally applied in the predication on soil water content, the solution of this macro-scale mathematical model takes a long time to execute. The precision of computation sometimes is reduced in order to save on calculation time. Therefore, the main objectives of this project are (i) to develop a parallel and distributed algorithm for the Soil Water Simulation Movement Model; (ii) implement the Soil Water Movement Simulation model on the Spider II

distributed system; (ii) to evaluate the performance of simulating the Soil Water Movement Model on Spider II.

1.3 Organization of the Documentation

The remaining sections of this documentation will be organized as follows: Chapter 2 describes the software requirement specification (SRS). The SRS follows the IEEE recommended practice for SRS (IEEE Std 830-1998). Chapter 3 provides a detailed description of the software architecture and design. Chapter 4 shows the test runs of Spider II for the Soil Water Movement Simulation Model and the performance evaluation. Chapter 5 contains the maintenance documentation. Finally, Chapter 6 draws a conclusion and points to further direction of the project.

CHAPTER TWO
SOFTWARE REQUIREMENTS
SPECIFICATION

2.1 Introduction

This Software Requirement Specification is for the simulation of soil water movement model using Spider distributed system. It will be the Master project of Li Wang for the degree of M. S. in Computer Science department, CSUSB.

2.1.1 Scope

The second version of Spider project (Spider II) was developed by Koping Wang [3] in his Master's thesis in 1999, which implemented the various features of Spider system — registry service, task management, load balancing and distributed computing service. The first version was developed by Han-Sheng Yuh [2].

This project will develop an application for agriculture, hydrology and environment science, which quantitatively describes and predicts the spatial and temporal changes of soil water content with a mathematical model called the Soil Water Movement Simulation Model. Because prediction and understanding of soil water movement

play an important role in designing and testing various irrigation schemes, there are many scientists in agriculture, hydrology and environment science who are working on calculating the model. The problem is that calculating the mathematical equations on a single PC or workstation will take a long time. The precision of computation sometimes is reduced in order to save calculation time. For this project, Spider system will improve the calculation speed for simulating soil water movement in time and space, and will generate dynamic 2D graphics to visualize the output.

This project will also implement several Java-based user interfaces and views to monitor the execution of Spider applications, which can allow users to access Spider system from any platform where there is a Web browser.

2.1.2 Overview

The rest of this software requirement specification will contain the following: first we overview the Spider system, then we specify the new application of Spider system.

2.2 Overall Description

Spider project is the first distributed computation research project in the Department of Computer Science at CSUSB. It is initially developed by Han-Sheng Yuh in his Master's thesis in 1997. It is an object-oriented distributed system. In his thesis, Han-Sheng introduced the overall goal for Spider and he defined major components for the system: Object Service Broker, Registry Server and Task Manager.

The second version of Spider developed by Koping Wang in 1999 (Spider II) continued to improve Spider features and additional new features, which include:

Load Balancing: The Registry Server will auto-detect Object-Servers whether they are up or down and check the percentage of utilization of the CPU usage. After detection, Registry will have a list of available servers and the priority of every Object-Server that the Registry Server suggests Task Manager to use.

Stabilizing the Multi-Tasking Operation: Spider II added a new mirror Registry Server. If the Registry Server goes down during the computation, the mirror will pick up the duty.

Ease of Adding New Service: Since Spider II is designed in the object-oriented paradigm, it will be easy to add new functions or services in the future. Developers just need to use the design patterns and utility programs to write a new distributed application.

2.2.1 Introduction to Model

More than 80% of total available water resource is being used in agriculture to meet huge food requirements of an expanded population. Water shortage problem is becoming much more serious all over the world. Optimal water management is vital for higher yields and to increase water use efficiency and minimize water quality degradation.

Because prediction and understanding the dynamics of soil water movement processes can guide farmers and policy makers in making decisions that optimize the dual goal of high crop yield, high water use efficiency and low environmental degradation, a simulation model was developed to quantitatively describe and predict soil water spatial and temporal movement after irrigation.

The model generally includes two parts – the partial differential equation and auxiliary conditions. The differential equation accurately described the dynamic process of water transport in soil. The auxiliary

conditions are the initial soil water content before irrigation and the boundary conditions around the soil area considered after irrigation. Figure 2 illustrates a numerical model describing the soil water movement in two coordinates (X and Z).

$$\left\{ \begin{array}{l} \frac{\partial \theta}{\partial t} = \frac{\partial}{\partial x} \left[D(\theta) \frac{\partial \theta}{\partial x} \right] + \frac{\partial}{\partial z} \left[D(\theta) \frac{\partial \theta}{\partial z} \right] - \frac{\partial K(\theta)}{\partial z} \\ \theta = \theta_a \quad t = 0 \quad z \geq 0 \quad 0 \leq x \leq L_h \end{array} \right. \quad [1] \quad [2]$$

where θ Soil Water Content ($\frac{cm^3}{cm^3}$)
 θ_a Initial Soil Water Content ($\frac{cm^3}{cm^3}$)
 $K(\theta)$ Hydraulic Conductivity ($\frac{cm^2}{min}$)
 $D(\theta)$ Soil Water Diffusivity ($\frac{cm^2}{min}$)
 t Time (minute)
 x Horizontal Coordinate Value
 z Vertical Coordinate Value

Figure 2. Two-Dimensional Water Movement Model

Eq. [1] is the differential equation, and Eq. [2] is the auxiliary conditions about the initial soil water content. After the differential equation and auxiliary conditions have been specified, the model is ready for

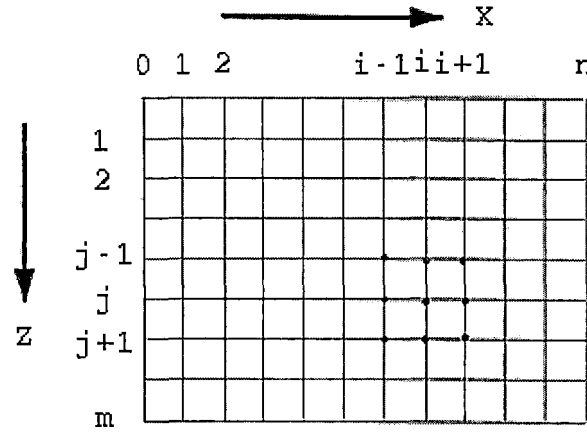


Figure 3. The Finite Difference Grid
for Two-Dimensional Space

computation to simulate water spatial and temporal movement.

The numerical solution of the partial differential equation in the model is based on the finite difference method using space and time discretization. For the considered soil profile (Figure 3), a grid containing (i, j) nodes in the direction of coordinate axes x, z , respectively, will be created. The corresponding difference equations resulting from the differential equation in the model (Figure 2) will be applied to every node of the grid and we shall have a system of finite difference equations (FDE) in the following form:

$$a_{ij}\theta_{i-1,j}^{k+1} + b_{ij}\theta_{i,j}^{k+1} + c_{ij}\theta_{i+1,j}^{k+1} = h_{i,j}$$

Where $1 \leq i \leq n$, $1 \leq j \leq m$, a_{ij} , b_{ij} and c_{ij} are coefficients, h_{ij} is a known value, $\theta_{i,j}^{k+1}$ represents soil water content at the point (i, j) in the grid for the current time period $k+1$ (assuming that the last time period is k). Using the known initial and boundary conditions, we can first calculate the soil water content at every point along the x-axis, $\theta_{1,1}^{k+1}, \theta_{2,1}^{k+1}, \dots, \theta_{i,1}^{k+1}, \dots, \theta_{n-1,1}^{k+1}$, then successively compute the $\theta_{i,j}^{k+1}$ along the z-axis at $j=2$, $j=3$, ..., $j=m-1$. For this time period, the difference method to solve the differential equation in the model is that the implicit difference method is used on x-axis and the explicit difference method is used on z-axis. When soil water contents at every point in the grid during $k+1$ time period ($\theta_{i,j}^{k+1}$) have been finished to calculate, then the computation of water contents at every point for the next time period, $k+2$, will be continued. But, for this time period, we first calculate the values along the z-axis, not x-axis, $\theta_{1,1}^{k+2}, \theta_{1,2}^{k+2}, \dots, \theta_{1,j}^{k+2}, \dots, \theta_{1,m-1}^{k+2}$, then successively compute the $\theta_{i,j}^{k+2}$ along the x-axis at $i=2, i=3, \dots, i=n-1$, because the explicit and implicit difference methods have been changed to use on the x-axis and z-axis, that is, the explicit difference method is used on x-axis and the implicit

difference method is used on z-axis. The water contents of every point of next period are calculated according to the computed results from the last period. This method to solve the FDE is called the alternating-directions implicit (ADI) difference method, which is highly efficient in solving problems of water distribution in two-dimensional rectangular space.

Because coefficients a_{ij} , b_{ij} and c_{ij} are calculated according to the soil water diffusivity (D) and the computation of D is dependent on the water content (θ), the system of FDE equations are nonlinear. In order to linearize the equations, the iterative method of Newton will be used, which the iterative computing process used for solving the equations is continued until

$$\text{MAX} \left| \frac{\theta_{i,j}^p - \theta_{i,j}^{p-1}}{\theta_{i,j}^{p-1}} \right| \leq \varepsilon$$

where $\varepsilon > 0$, is a prescribed error tolerance, and p is the test computation times. This method makes the solution both computationally simple and efficient, despite the nonlinear dependence of D on θ .

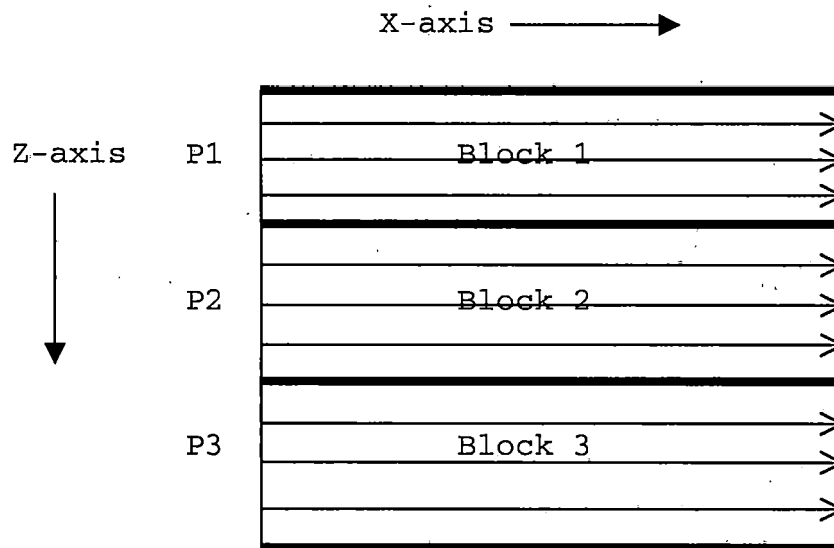
2.2.2 Product Perspective

Although the numerical model has been widely used in the predication on soil water content, the solving the macro-scale mathematical models will take long time. The precision of computation sometimes is reduced in order to save calculation time. For example, the step length of x- or z-axis may be too large to accurately simulate the water movement in the practical system. Or the time step between two time periods is so big that we can not see the instant changes of soil water content. In this project, the Soil Water Movement Simulation Model will be implemented on the Spider to improve calculation speed and precision for solving the model.

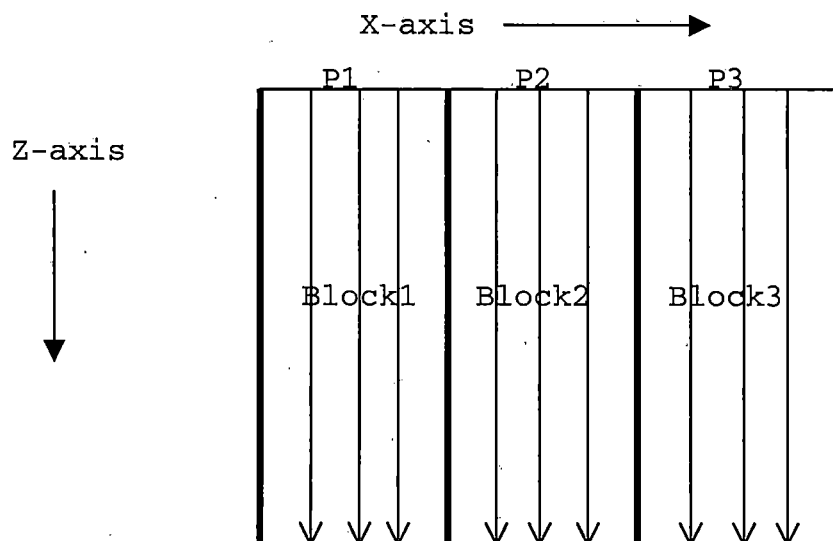
2.2.2.1 The Distributed Algorithm. The Figure 4 presents an example of data distribution, which is for 3 processors and two time periods.

The simulation space grid is divided into several equal size blocks. The computation of every node in these blocks will be distributed to the processors at the same time. Each processor calculates one block at one time (Figure 4). When calculating the values for the first time period after irrigation, the space grid is divided along

the horizontal direction. Thereafter, during the second time period after irrigation, the space grid is



The first time period



The second time period

Figure 4. The Distributed Algorithm

divided along the vertical direction. The calculation for the next time period is dependent on the results from the last time period. When the processors finished the computation for the last time period, they continue to calculate for the next time period. Finally, all nodes of the grid in different time periods can be successively calculated.

2.2.2.2 System Interface. In the Spider system, the computation will be done in the following steps:

1. Sending service requests to OSB, then OSB ask the Task Manager for the task ID and the Server-Objects (processors).
2. Task Manager will issue the task ID and a list of Object-Servers that OSB can use. Then OSB will distribute the blocks to the Object-Servers. After the Object-Servers finish the computation for the last time period, OSB will collect the result and send back to the Object-Servers to continue calculating for the next time period after irrigation.
3. When the computation for time period has been finished, OSB will return the task ID to the Task Manager.

Figure 5 shows the deployment diagram for this project.

2.2.2.3 User Interface. The user interface of this software will be done using Java language and html. Users can use any computer in the Computer Science Net or the

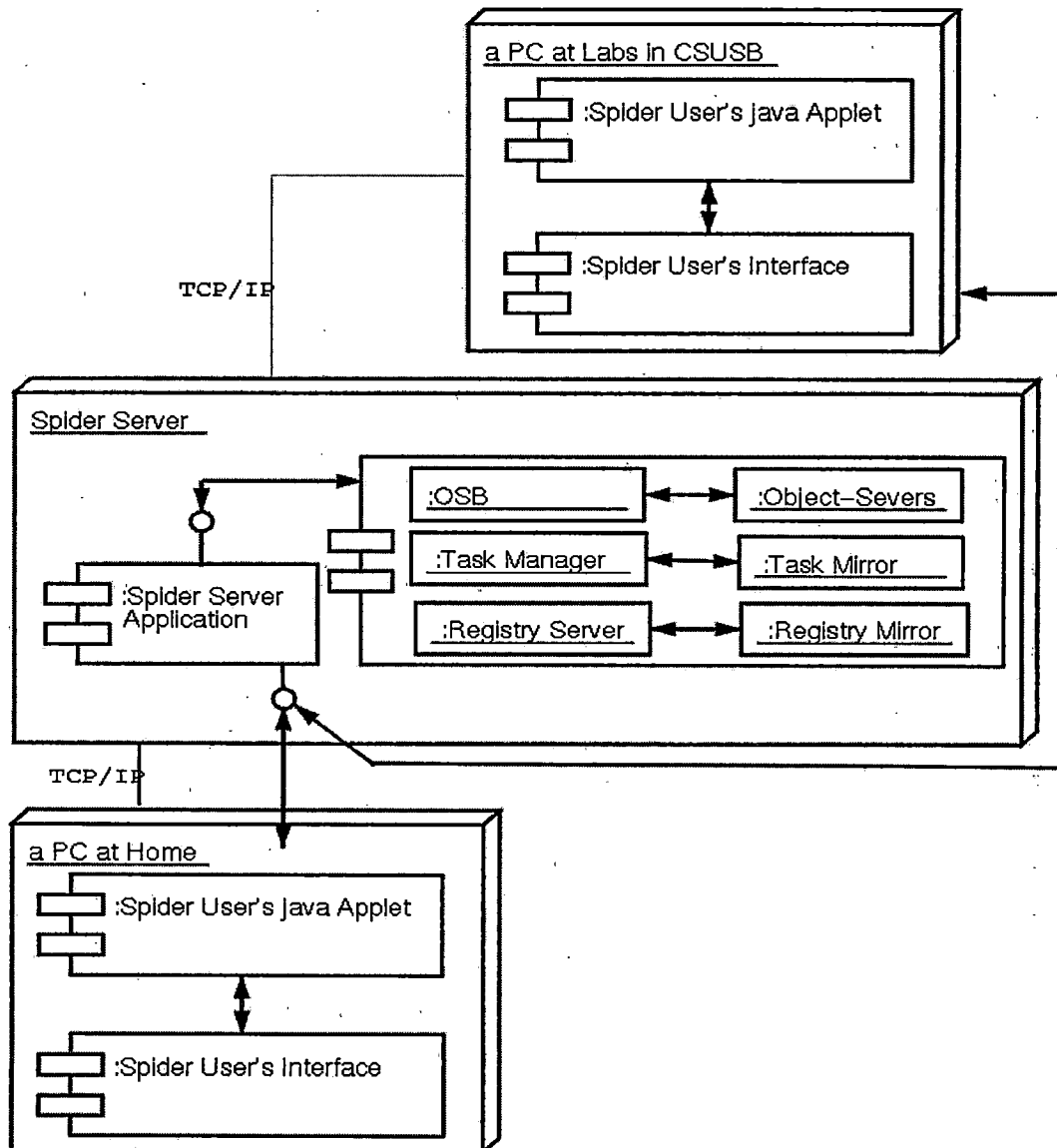


Figure 5. Deployment Diagram

Spider Server, which has Internet access and Web browser, to run this software.

2.2.2.4 Hardware Interfaces. If run on the Computer Science Net, there is no requirement for hardware. If run at home, the workstation should be at least CPU 500 MHz, RAM 160 MB, and Modem 56 KB.

2.2.2.5 Software Interfaces. The software required to develop this software are following:

- Spider II system
- Linux RedHat 7.0
- GNU C++ programming for Linux

2.2.2.6 Communication Interfaces. Ethernet 100 base network connection for Linux box is used.

2.2.2.7 Operations. The operations of this software will be the following: First, start the Spider system, then run this application. This software will prompt users to input the initial conditions which accomplish the software requirements.

This software will only run during scheduled periods or by request. It will not run 24 hours a day, 7 days a week.

2.2.3 Product Functions

Shown on Figure 6 is the Use Case Diagram of Soil Water Movement Model Software:

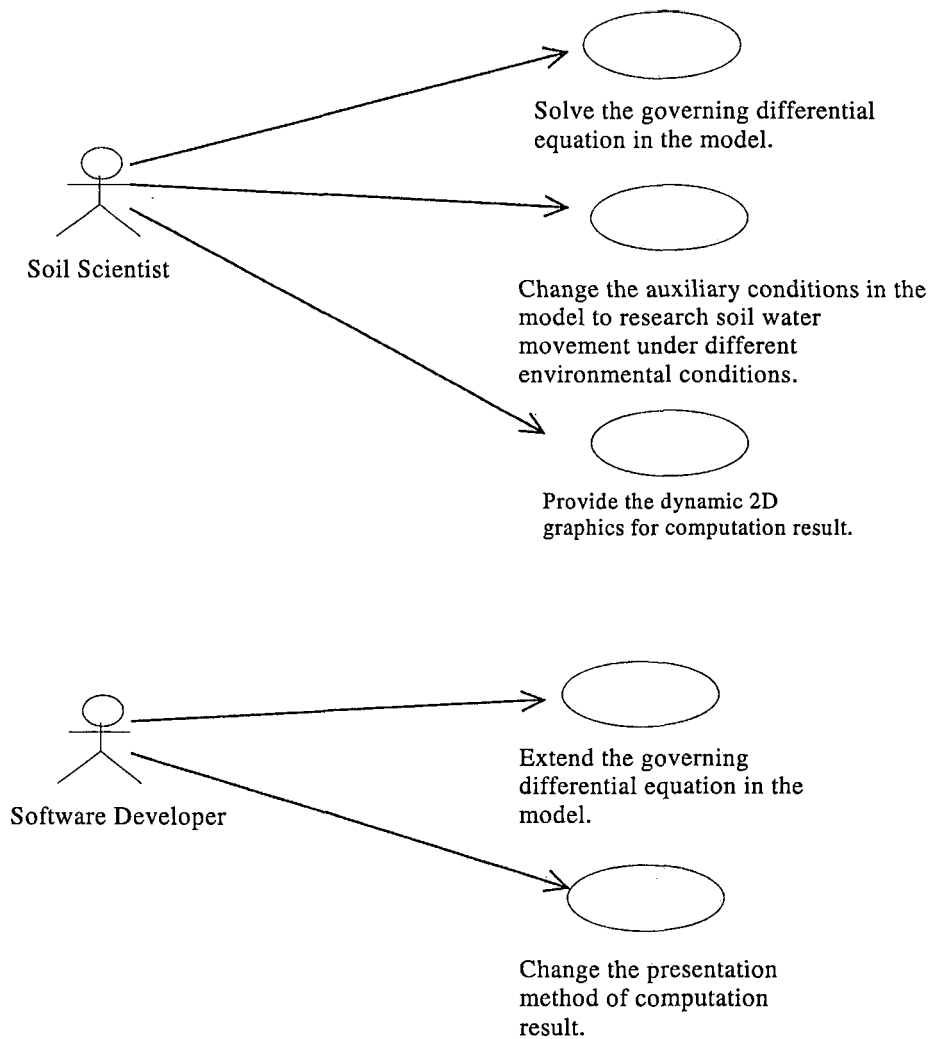


Figure 6. User Case Diagram

For the soil scientists:

Solve the governing differential equation: This function is to solve the differential equation ([1]), see in Figure 2 Soil scientists do not need to know how to implement the differential equation in the software and can directly use it.

Change the auxiliary conditions: It is a main research area for soil scientists to change the expression of the auxiliary conditions according to different environmental conditions and to simulate soil water movement in the practical system. Because this software will be designed in the object-oriented paradigm, it will be easy to change or add any auxiliary conditions in future. This function will play an important role in research and management on water flow and solute transport in soils.

Provide the dynamic 2D graphics: The dynamic 2D graphics of soil water contents in time and space will be generated to visualize the computation result using java language. The graph will present the iso-water content curves.

For software developers:

Extend the governing differential equation in the model: The differential equation ([1]) in Figure 2 is

widely used in various research areas. For a special case, it is possible that the differential equation can add new terms. Because this software will be designed in the object-oriented paradigm, it will be easy to add new functions in future and to extend the differential equation.

Change the presentation method of computation results:

Because the computation result is stored in a data file, the software developers can easily change the presentation method of computation results according to the soil scientist's requirements.

2.2.4 User Characteristics

The users of this software (soil scientists) does not need to know about programming if they do not want to change the auxiliary conditions in the model. But if the user would want to change the expression of the auxiliary conditions, they just need to know how to change the functions about auxiliary conditions using C++ language.

All developers need to have experience with C++ and Java programming under Unix environment, understand network programming concept and object-oriented Paradigm in order to extend this application according to the soil scientist's requirements

2.2.5 Constraints

This software will only run on the Spider System in the computer science network.

The platform will only support UNIX or UNIX-like operating system. Linux will be the primary environment during development.

2.3 Specific Requirements for External Interface

For the Soil Water Movement Simulation Model application, several Java-based user interface will be implemented based on the user's requirements. They will be able to run on any platform, where there is a Web browser. Therefore, anyone who is an authorized user is able to access the Spider's distributed computing system through Internet to run the simulation of the Soil Water Movement Model.

The window layouts for user interfaces are enumerated in the Appendix. They include:

1. Welcome window

It introduces users into the Spider project.

2. Soil Water Movement Model window

It introduces the Soil Water Movement Model.

3. Initial Condition window

It prompts users to input the initial conditions which are required by the application.

4. Transferring Files window

It allows users to transfer files to and from a remote computer (Spider Server), and work with files and directories on that remote computer.

5. Monitoring and Displaying window

It monitors the execution of the application on the Spider system and views the dynamic 2D graphical Display of the computation results.

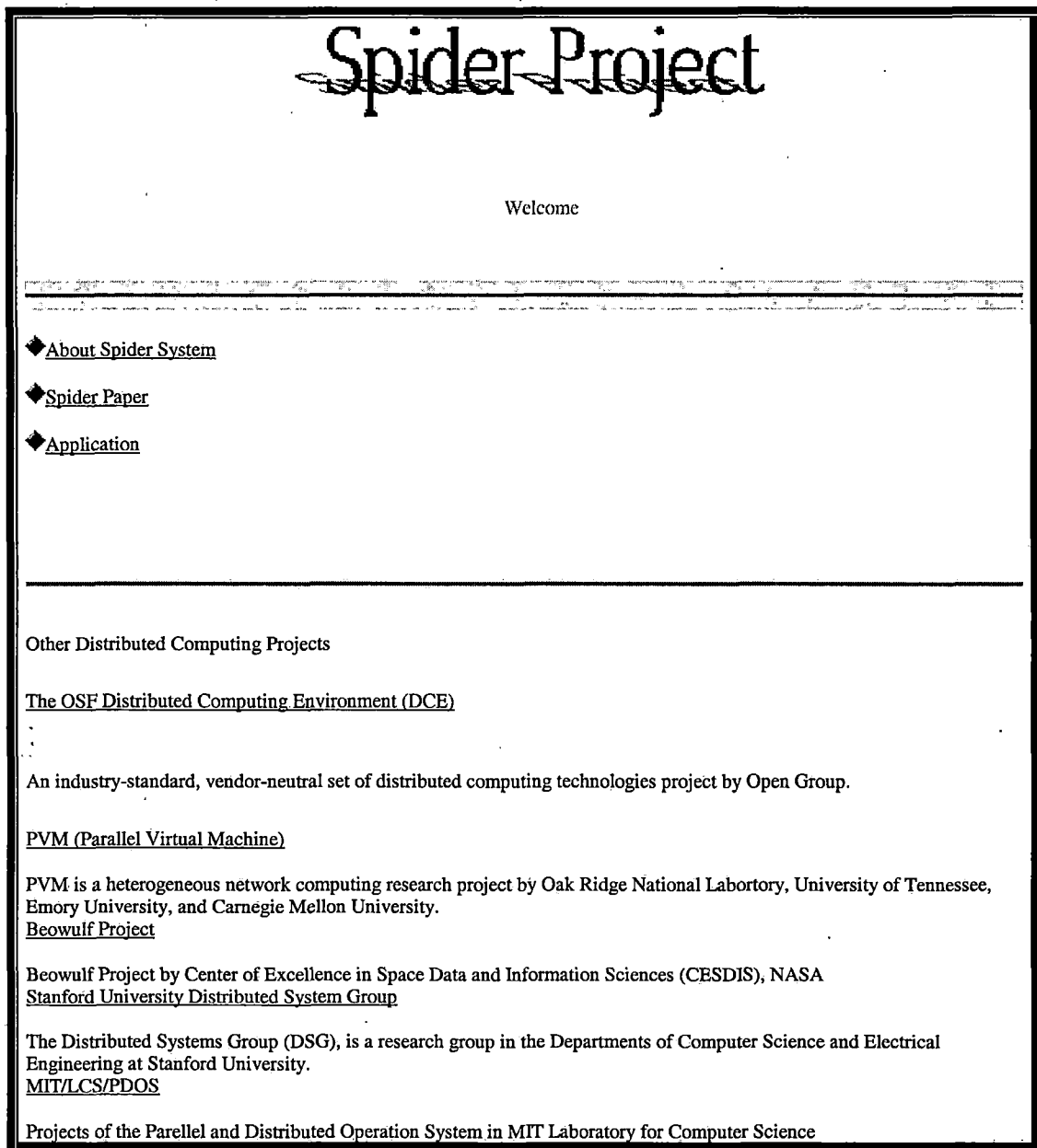



Figure 7. Welcome Web Page

This window introduces the users into the Spider project. For the Soil Water Movement Model application, clicking the "Application" button.



Simulation of Soil Water Movement Model

This application is for agriculture, hydrology and environment science, which quantitatively describes and predicts the spatial and temporal changes of soil water content with mathematical model. Because prediction and understanding of soil water movement play an important role in designing and testing various irrigation schemes, the simulation of soil water movement model has been widely used in the predication on soil water content. But the solving the macro-scale mathematical models will take much time on PC. This application will be implemented on the Spider II to improve calculation speed for solving the model.

This application uses the following numerical model which describes the soil water movement in two coordinates (x and z):

$$\frac{\partial \theta}{\partial t} = \frac{\partial}{\partial x} \left[D(\theta) \frac{\partial \theta}{\partial x} \right] + \frac{\partial}{\partial z} \left[D(\theta) \frac{\partial \theta}{\partial z} \right] - \frac{\partial K(\theta)}{\partial z}$$

where

θ Soil Water Content

$K(\theta)$ Hydraulic Conductivity

$D(\theta)$ Soil Water Diffusivity

t Time

x Horizontal Coordinate Value

z Vertical Coordinate Value

[start this application on Spider system](#)

[\[back\]](#)

Figure 8. Introduction Web Page for Model

Please enter the initial conditions of the simulation:

The soil vertical depth (cm)

The soil horizontal distance (cm)

The step on the vertical direction (cm)

The step on the horizontal direction (cm)

The time length after starting irrigation (minutes)

The time step (minutes)

The prescribed error tolerance

The initial soil water content before irrigation (cm^3/cm^3)

The boundary soil water contents after irrigation (cm^3/cm^3)

1. top

2. bottom

3. left

4. right

The buried vertical depth for water source (cm)

The saturated water content (cm^3/cm^3)

The file name for this initial conditions

Figure 9. Web Page to Input the Initial Condition

This window prompts users to input the initial condition which are required by this application:

The first field is used to input *the soil vertical depth*.

The second field is used to input *the soil horizontal distance*.

The third field is used to input *the step on the vertical direction*.

The forth field is used to input *the step on the horizontal direction*.

The fifth field is used to input *the time length after starting irrigation*.

The sixth field is used to input *the time step*.
The seventh field is used to input *the prescribed error tolerance*.

The eighth field is used to input *the initial soil water content before irrigation*.

The ninth, tenth, eleventh and twelfth fields are used to input the boundary soil water contents after irrigation at top, bottom, left and right respectively.

The thirteenth field is used to input *the buried vertical depth for water source*.

The fourteenth field is used to input *the saturated water content*.

When finishing to input the initial conditions, the file name for the initial conditions need to be given and saved, then submit it.



Monitoring and Displaying

During the distributed computation process, this window provides several animated views to monitor the execution of Spider programs:

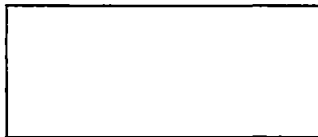
- Network View displays the machines selected by the Spider system and each host is represented by an icon image showing host name.
 - Task View shows status of all tasks as they execute across all hosts and each task is represented by a horizontal bar along a common time line axis.
 - Application View concurrently displays the 2D graphic of computation result during the distributed computation process.
- After the distributed computation,
- ◆ The dynamic 2D graphic can be displayed by clicking "Dynamic 2D Graphic Display of Results" button. The time after starting irrigation for 2D graphic can automatically display in the "Time After Starting Irrigation" box. It can be paused by clicking "Pause" button.
 - ◆ If you want to look at 2D graphic at some time point, please pause the 2D graphic, then enter the time into the "Time After Starting Irrigation" box.
 - ◆ If you want to get the output file of computation results from Spider Server to your local file system, please click the "Get the Output File of Computation results" button.
 - ◆ If you want to revise the initial condition file and restart the computation, please click the "Revise the Initial Conditions" button.

Task:

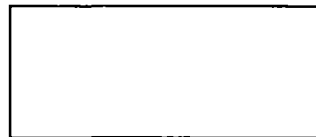


(please input the executable file name and the number of machines that will be used, for example, water flow 15)

Network View



Task View

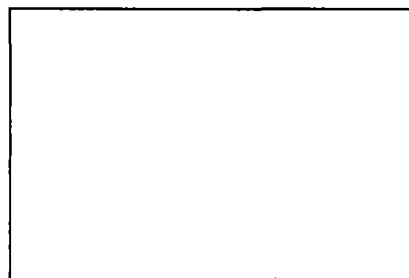


Colour Legend for Soil Water Content

($\text{cm}^3 / \text{cm}^3$)

	0.08
	0.12
	0.16
	0.20
	0.24
	0.28
	0.32
	0.36
	0.40
	0.44

Application View



[Dynamic 2D Graphic Display of Results]

Time After Irrigation

stop

continue

[Get the Output file of Computation Results]

Figure 10. Web Page for Monitoring and Displaying

This Window monitors the execution of this application on the Spider system and views the dynamic 2D graphical Display of computation results.

CHAPTER THREE

SOFTWARE DESIGN

The software development used in implementing Soil Water Movement Model (SWaMM) is the object-oriented approach. The design methodology utilized the Unified Modeling Language (UML), which simplified the complex process of software design by making a "blue print" for the software construction.

First, we discuss the distributed algorithm on how the SWaMM will be simulated on Spider II. Second, we discuss the software architecture. And lastly, we show the detailed design in pseudo-code.

3.1 Distributed Algorithm

The UML deployment diagram shows the physical components and physical communication protocols between various components of the software system, see Figure 5.

When a client PC station at home or lab in the CS Net requests a distributed service, for example, computation of SWaMM, from user's interface implemented in Java applet on the Web browser, then Spider server receives and sends it to the master machine where OSB is running in the Spider system. There is a daemon on the master machine that

receives the message from the Spider server. After that, the master machine will activate the water movement simulation to start the distributed computation on the Spider system.

In the Spider system, OSB that is located on the master machine will be started to communicate with the Task Manager for task ID and the list of available Object-Servers by using TCP socket. OSB also distributes data to the available Object-Servers. TCP is applied between Registry Server and Task Manager as well.

During computation, the results will be sent to the Spider Server to draw the dynamic 2D diagram to be viewed by user from Web browser. The status of computation of this application on the Spider system is displayed on the user's Web browser as well. After completing all the computation, OSB will return the task ID to the Task Manager. If Registry Server or Task Manager crashes during computation, the mirror will pick up the duty.

SWaMM (Figure 2) consists of two parts — the partial differential equation and auxiliary conditions. The partial differential equation calculates the dynamic process of water transport in a two-dimensional soil profile basing on

the auxiliary conditions input by user as the initial values at the beginning of computation. The auxiliary conditions are the initial soil water contents before irrigation and the boundary conditions around the considered soil area after irrigation.

The numerical solution of the partial differential equation is based on the finite difference method by space and time discretization. So first, we need to create a finite difference grid for a considered space (Figure 3). The distance steps of X and Z directions in the grid are given by the user as the initial conditions besides the finite depth and horizontal distance, which determines the number of nodal points in the grid to be calculated, and it will influence the accuracy of solution of partial differential equation. The more accuracy the user wants, the more time the computation takes, because there are more nodal points to be calculated. Calculating the water contents of every nodal points on the considered grid at every time level until the end of the simulating time required by users is to be implemented.

The implicit difference method which is called a "backward-difference approximation" [5] relative to the $k+1$ time level at which the space differences are expressed,

and explicit difference method which is called "forward-difference scheme" [5] relative to time k level, are used to solve the partial differential equation of the SWaMM. These two methods are alternatively applied on the horizontal and vertical directions, that means if an explicit difference method is used on the vertical direction and the implicit difference method on the horizontal direction at this time level k, then for the next time level k+1, those two difference methods will be exchanged to apply on two directions, which is called the alternating directions implicit (ADI) difference method. ADI method is highly efficient in solving problems of water distribution in two-dimensional rectangular space. How the partial differential equation to be solved into difference equations using the ADI method is illustrated in the following:

1. Applying implicit difference method to the horizontal direction and explicit difference method to the vertical direction. The items of the partial differential equations can be approximated to be:

$$\frac{\partial \theta}{\partial t} \approx (\theta_{i,j}^{k+1} - \theta_{i,j}^k) / \Delta t$$

$$\frac{\partial}{\partial x} \left[D(\theta) \frac{\partial \theta}{\partial x} \right] \approx \left[D_{i+\frac{1}{2},j}^{k+1} (\theta_{i+1,j}^{k+1} - \theta_{i,j}^{k+1}) - D_{i-\frac{1}{2},j}^{k+1} (\theta_{i,j}^{k+1} - \theta_{i-1,j}^{k+1}) \right] / \Delta x^2$$

$$\frac{\partial}{\partial z} \left[D(\theta) \frac{\partial \theta}{\partial z} \right] \approx \left[D_{i,j+\frac{1}{2}}^k (\theta_{i,j+1}^k - \theta_{i,j}^k) - D_{i,j-\frac{1}{2}}^k (\theta_{i,j}^k - \theta_{i,j-1}^k) \right] / \Delta z^2$$

$$\frac{\partial K(\theta)}{\partial z} \approx (K_{i,j+1}^k - K_{i,j-1}^k) / 2\Delta z$$

$$\text{and } r_1 = \frac{\Delta t}{\Delta x^2}, \quad r_2 = \frac{\Delta t}{\Delta z^2}, \quad r_3 = \frac{\Delta t}{2\Delta z}$$

So the partial differential equation of SWaMM can be rewritten to be the following difference equation according to the above difference results:

$$a_{i,j} \theta_{i-1,j}^{k+1} + b_{i,j} \theta_{i,j}^{k+1} + c_{i,j} \theta_{i+1,j}^{k+1} = h_{i,j} \quad (3.1)$$

where

$$a_{i,j} = r_1 D_{i-\frac{1}{2},j}^{k+1} \quad (3.2)$$

$$b_{i,j} = - \left[1 + r_1 \left(D_{i-\frac{1}{2},j}^{k+1} + D_{i+\frac{1}{2},j}^{k+1} \right) \right] \quad (3.3)$$

$$c_{i,j} = r_1 D_{i+\frac{1}{2},j}^{k+1} \quad (3.4)$$

$$\begin{aligned} h_{i,j} = & -r_2 D_{i,j-\frac{1}{2}}^k \theta_{i,j-1}^k + \left[r_2 \left(D_{i,j-\frac{1}{2}}^k + D_{i,j+\frac{1}{2}}^k \right) - 1 \right] \theta_{i,j}^k \\ & - r_2 D_{i,j+\frac{1}{2}}^k \theta_{i,j+1}^k + r_3 (K_{i,j+1}^k - K_{i,j-1}^k) \end{aligned} \quad (3.5)$$

Using the given initial and boundary soil water contents and solving the algebraic equation group

formalized from (3.1) equation (addressed in the next section 3.3), the water contents $\theta_{1,1}^{k+1}, \theta_{2,1}^{k+1}, \dots, \theta_{i,1}^{k+1}, \dots, \theta_{n-1,1}^{k+1}$ will be calculated along the horizontal direction (X) at $j=1$ on the vertical direction, then successively compute at $j=2, \dots, m-1$. After finishing to calculate all the water contents of every nodal point at this time level, then advance the solution to the next time level using the following difference equation.

2. Applying implicit difference method on the vertical direction and explicit difference method on the horizontal direction. The items of the partial differential equations can be approximated to be:

$$\frac{\partial \theta}{\partial t} \approx (\theta_{i,j}^{k+1} - \theta_{i,j}^k) / \Delta t$$

$$\frac{\partial}{\partial x} \left[D(\theta) \frac{\partial \theta}{\partial x} \right] \approx \left[D_{i+\frac{1}{2},j}^k (\theta_{i+1,j}^k - \theta_{i,j}^k) - D_{i-\frac{1}{2},j}^k (\theta_{i,j}^k - \theta_{i-1,j}^k) \right] / \Delta x^2$$

$$\frac{\partial}{\partial z} \left[D(\theta) \frac{\partial \theta}{\partial z} \right] \approx \left[D_{i,j+\frac{1}{2}}^{k+1} (\theta_{i,j+1}^{k+1} - \theta_{i,j}^{k+1}) - D_{i,j-\frac{1}{2}}^{k+1} (\theta_{i,j}^{k+1} - \theta_{i,j-1}^{k+1}) \right] / \Delta z^2$$

$$\frac{\partial K(\theta)}{\partial z} \approx (K_{i,j+1}^{k+1} - K_{i,j-1}^{k+1}) / 2\Delta z$$

So the partial differential equation of water movement model can be rewritten to be the following differential equation according to the above difference results:

$$a'_{i,j}\theta_{i,j-1}^{k+1} + b'_{i,j}\theta_{ij}^{k+1} + c'_{i,j}\theta_{i,j+1}^{k+1} = h'_{i,j} \quad (3.6)$$

where

$$a'_{i,j} = r_2 D_{i,j-\frac{1}{2}}^{k+1} \quad (3.7)$$

$$b'_{i,j} = -\left[1 + r_2 \left(D_{i,j-\frac{1}{2}}^{k+1} + D_{i,j+\frac{1}{2}}^{k+1}\right)\right] \quad (3.8)$$

$$c'_{i,j} = r_2 D_{i,j+\frac{1}{2}}^{k+1} \quad (3.9)$$

$$h'_{i,j} = -r_1 D_{i-\frac{1}{2},j}^k \theta_{i-1,j}^k + \left[r_1 \left(D_{i-\frac{1}{2},j}^k + D_{i+\frac{1}{2},j}^k\right) - 1\right] \theta_{i,j}^k - r_1 D_{i+\frac{1}{2},j}^k \theta_{i+1,j}^k + r_3 \left(K_{i,j+1}^{k+1} - K_{i,j-1}^{k+1}\right) \quad (3.10)$$

Using the given initial and boundary soil water contents and solving the algebraic equation group formalized from (3.6), the water content $\theta_{1,1}^{k+2}, \theta_{1,2}^{k+2}, \dots, \theta_{1,j}^{k+2}, \dots, \theta_{1,m-1}^{k+2}$ will be first calculated along the vertical direction (Z) at $i=1$, then successively compute at $i=2, \dots, i=n-1$.

According to (3.2), (3.3), (3.4) and (3.5), we can know there are not data dependences between the two neighbor nodal points on horizontal direction in space grid, and according to (3.7), (3.8), (3.9) and (3.10), we can find there are not data dependences between the two neighbor nodal points on vertical direction, so the distributed protocol for computation of SWaMM is illustrated in Figure 4.

The space grid is horizontally divided and distributed at an odd time level and vertically divided and distributed at the next even time level. The computation of water contents, which is iteratively distributed on horizontal and vertical direction at two different continued time levels, will be lasted to the simulating time of irrigation required by user as the initial condition.

Because this distributed computation of SWaMM mixes input, calculation and output, that is, during one time, data need to be sent, calculated and returned, the program creates several threads to deal with input, calculation and output respectively. On the master side, we create two kinds of threads to be separately responsible for sending data and receiving data that are named *sender* and *getter*. And the number of threads *getter* is same as the number of slaves used, that is, every slave has its own *getter* thread on the master side. On every slave side, there are only three threads to handle the receiving, calculating and returning data that are named *receiver*, *calculator* and *returner*.

In order to further improve the performance, every block in Figure 4 is divided into several parts again, which results just one part of every block is sent to its

slave every time, actually not the whole block of data. This makes *calculator* can start to calculate as soon as *receiver* just receives a part of data and the *returner* returns this part of calculated results at once, and the *getter* on the master machine can get this part of results while the *sender* still send the data to slaves. This work state of every machine in Spider system looks like a continuous cycle with starting at thread *sender* as showed in Fig.3.1.

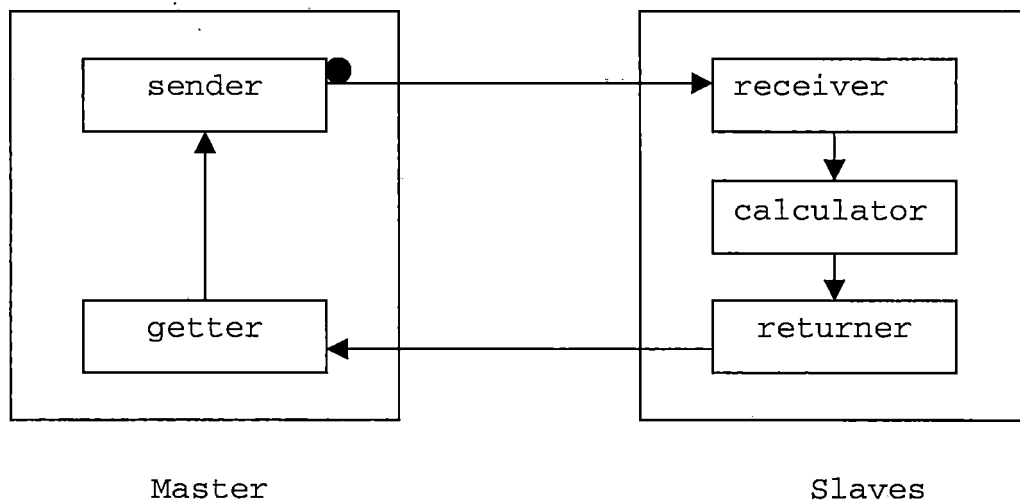


Figure 11. Work Cycle of Threads

So the distribution and calculation protocol of this software is as the following:

1. The simulation space grid is divided into several block which is going to distributed to the slaves and

every block is continually split to several parts when starting computation. This program distributes the parts to the size that is enough for slaves to calculate two lines of data every time.

2. The thread *sender* on master creates several processors by *fork* system call whose number is same as the number of slaves. These processors send data to various slaves at the same time and send only one part of their corresponding blocks every time.
3. Once thread *receiver* on slave receives one part of data, the thread *calculator* starts calculation at once, and thread *returner* also returns the calculated data as soon as possible. So *receiver*, *calculator* and *returner* separately receives, calculates and returns data part by part till the whole block of data.
4. The threads *getter* get the calculated results from the slaves part by part while the thread *sender* still sends data to slaves. As soon as the threads *getter* have received the part of data, they start to rearrange these data on the different direction preparing for the computation of the next time level. After threads *getter* have finished receiving their whole block of calculated results, then they informs

the thread sender to send these just received data again.

5. The steps 1-4 are repeated and the whole computation of this software is like a loop increased by one time step till to the simulating time of irrigation required by user, while every computation for next time level is depend on the results from the last time. At every given interval time during the computation, the water content results of the whole soil profile will be automatically translated to be coordinate values and sent to Spider server to draw 2D diagram which is viewed on web browser with java-based user interface.

3.2 Software Architecture

All the class components of this software system are drawn using UML class diagram as showed in Figure 12.

3.2.1 Classes *WaterMovement* and

WaterMovement_slv

Classes *WaterMovement* and *WaterMovement_slv* are two important classes in the software design. Class *WaterMovement* is run on the master side, while *WaterMovement_slv* is run on the slave side. At the

beginning, the *WaterMovement* is activated by the user's distribution require and *WaterMovement* sends this require to task manager to ask for the list of available machines. After that, the communication sockets between slaves and master is built at once by the *Call_OSB_initialData()* function of class *WaterMovement* and *Start_OSB()* function of class *WaterMovement_slv*. When these sockets are ready to be used, it will not closed anytime till all the computation has been completely finished.

Class *WaterMovement* is used by thread *sender* to send data with the *send_data()* function and is used by thread *getter* to get data, change data and send data to Spider server to draw the 2D diagram using the *chang_data_to_draw()* function.

Class *WaterMovement_slv* is used by thread *receiver* on the slave to receive initial conditions and open memory space for some variables used in the program.

WaterMovement_slv is mainly used by thread *calculator* to calculate the $D_{i,j-\frac{1}{2}}^k$, $D_{i,j+\frac{1}{2}}^k$ in (3.5) by *calculate_preZD()*

function, $K_{i,j+1}^k$, $K_{i,j-1}^k$ in (3.10) by *calculate_preZK()*

function, $D_{i-\frac{1}{2},j}^{k+1}$, $D_{i+\frac{1}{2},j}^{k+1}$ in (3.2), (3.3), (3.4), (3.7), (3.8),

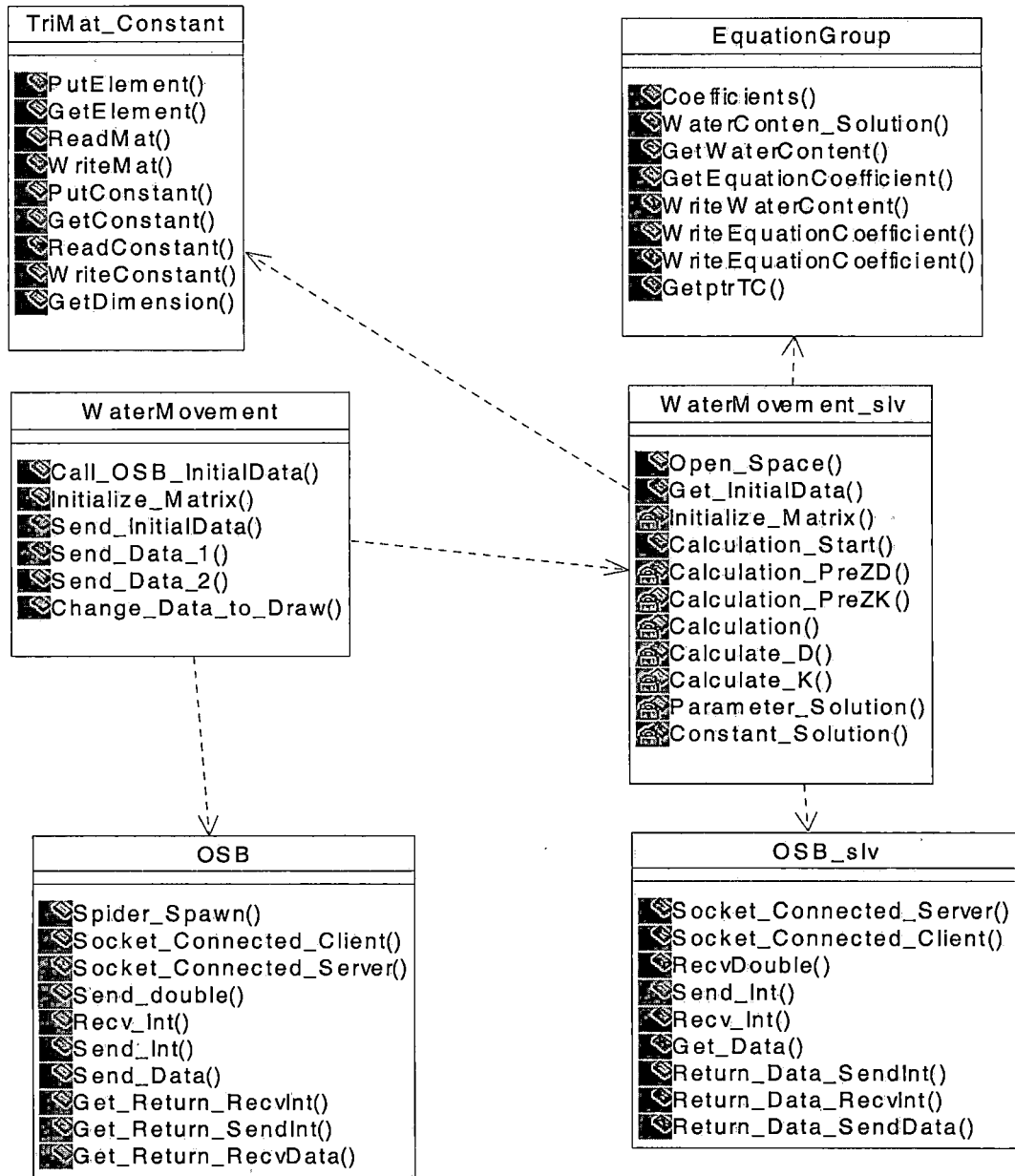


Figure 12. Component Architecture of Software

(3.9) by `calculate_D()` function, and $K_{i,j+1}^{k+1}$, $K_{i,j-1}^{k+1}$ in (3.10) by `calculate_K()` function. After calculating these results,

`parameter_solution()` function is used to calculate the $a_{i,j}$,

$b_{i,j}$ and $c_{i,j}$ or $a'_{i,j}$, $b'_{i,j}$ and $c'_{i,j}$, and `constants_solution()`

$$\begin{bmatrix} b_{1,j} & c_{1,j} & 0 & 0 & 0 \\ 0 & a_{2,j} & b_{2,j} & c_{2,j} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{n-2,j} & b_{n-2,j} & c_{n-2,j} \\ 0 & 0 & 0 & a_{n-1,j} & b_{n-1,j} \end{bmatrix} \begin{bmatrix} \theta_{1,j}^{k+1} \\ \theta_{2,j}^{k+1} \\ M \\ \theta_{n-2,j}^{k+1} \\ \theta_{n-1,j}^{k+1} \end{bmatrix} = \begin{bmatrix} h_{1,j} \\ h_{2,j} \\ M \\ h_{n-2,j} \\ h_{n-1,j} \end{bmatrix}$$

where $j=1,2,\dots,m$;

Odd Time Level

$$\begin{bmatrix} b_{i,1} & c_{i,1} & 0 & 0 & 0 \\ 0 & a_{i,2} & b_{i,2} & c_{i,2} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{i,m-2} & b_{i,m-2} & c_{i,m-2} \\ 0 & 0 & 0 & a_{i,m-1} & b_{i,m-1} \end{bmatrix} \begin{bmatrix} \theta_{i,1}^{k+1} \\ \theta_{i,2}^{k+1} \\ M \\ \theta_{i,m-2}^{k+1} \\ \theta_{i,m-1}^{k+1} \end{bmatrix} = \begin{bmatrix} h_{i,1} \\ h_{i,2} \\ M \\ h_{i,m-2} \\ h_{i,m-1} \end{bmatrix}$$

where $i=1,2,\dots,n$;

Even Time Level

Figure 13. Tridiagonal Matrix

function for $h_{i,j}$ or $h'_{i,j}$ to build the tridiagonal matrix to solve the algebraic equation group and finally find the water contents. The tridiagonal matrices are illustrated in Figure 13.

3.2.2 Classes *OSB* and *OSB_slv*

Classes *OSB* and *OSB_slv* form the communication and distribution functions for slave side and master side programs, respectively. *WaterMovement* class asks for distributed object-servers from the Task Manager by the *OSB*. Any communication between slave and master is archived by *OSB* running on master side and *OSB_slv* running on slave side. *socket_connect_server()* and *socket_connect_client()* functions are used to build sockets between slave and master. The total number of sockets built between slave and master is two times the number of slaves used in this computation. Half of these sockets are initialized for slaves as server and master as client, and the others are established for slaves as client and master as server.

3.2.3 Classes *TriMat_Constant* and

EquationGroup

Class *TriMat_Constant* is a utility for building the tridiagonal matrices with $a_{i,j}$, $b_{i,j}$, $c_{i,j}$ and $h_{i,j}$ or $a'_{i,j}$, $b'_{i,j}$, $c'_{i,j}$ and $h'_{i,j}$ resulted from *WaterMovement_slv* class. The *EquationGroup* class is another utility for solving the algebraic equation group basing on the tridiagonal matrices built by the *TriMat_Constant* class.

The Gaussian elimination is used to solve algebraic equation group, which includes two steps — forward elimination and back substitution.

1. Forward elimination

Assumed:

$$y_1 = \frac{h_1}{b_1}, \eta_1 = \frac{c_1}{b_1}$$

$$y_i = \frac{h_i - a_i y_{i-1}}{b_i - a_i \eta_{i-1}}$$

$$\eta_i = \frac{c_i}{b_i - a_i \eta_{i-1}}$$

$$(i=2, 3, \dots, n-2)$$

Then, eliminate matrices in Fig.3.3 with above coefficients to be the following equation group:

$$\theta_1 = y_1 - \eta_1 \theta_2$$

$$\theta_2 = y_2 - \eta_2 \theta_3$$

.....

$$\theta_i = y_i - \eta_i \theta_{i+1}$$

.....

$$\theta_{n-2} = y_{n-2} - \eta_{n-2} \theta_{n-1}$$

$$\theta_{n-1} = \frac{h_{n-1} - a_{n-1} y_{n-2}}{b_{n-1} - a_{n-1} \eta_{n-2}}$$

2. Back substitution

First, θ_{n-1} can be solved because h_{n-1} , a_{n-1} , y_{n-2} , b_{n-1} and η_{n-2} are given. Then solve θ_{n-2} , θ_{n-3} , ..., θ_3 , θ_2 , θ_1 successively from bottom to top with substitution method.

3.3 Detailed Design

This section discusses the detailed design of the five threads implemented in SWaMM — *sender* and *getter* threads running on the master side, as well as *receiver*, *calculator* and *returner* threads running on the slave side.

3.3.1 Thread *Sender*

The *sender* thread is the *main()* function in the *WaterMovement.C* on master side. Firstly, it declares an object of *WaterMovement* class and initialized it with the initial conditions input by the user via Java-based interface. Second, it creates the same number of *getter* threads as the number of slaves in order to make every slave to have its own *getter*. Third, it goes to a *for* loop which is increased by one time step and is finished at the end of the simulating time of irrigation required by user. There is a global variable *get_done* to check if all *getter* threads have finished to receive the whole block of

calculated results from their own slaves. If so, the computation goes to the next new time step.

This multithread program uses the *mutex* to synchronize the accesses to the global variables. To control accesses to a critical section of code, you lock a *mutex* before entering the code section and then unlock it when you have finished. The global variable *get_done* is used by thread *sender* and *getter* with *mutex*. If one of *getter* threads is done with getting its whole block of calculated results from its slave, the *get_done* will be increased by 1 by locking it with *mutex*. If the *get_done* value equals to the number of slaves, it means all the block of data in the whole soil profile have been received by the master and so thread *sender* can start to send data for computation of the next new time step again. So the thread *sender* needs to check *get_done* to see if *get_done* has increased to the number of slaves by locking it with *mutex* as well. If thread *sender* find *get_done* has gone up the number of slaves, it will continue, otherwise, it will wait by unlocking *get_done* and check it again later. Every time when a new thread *getter* has completed its receiving, it will notify and enforce thread *sender* to check *get_done*

again. The pseudo code of *sender* thread is showed in Figure 14.

```
Main()
{
    input initial conditions;
    call OSB to create sockets;
    create threads getter;
    for( i = 0; till the end of simulating time of irrigation;
        increased by one time step)
    {
        if( i = 0 )
            send data horizontally divided;
        else {
            lock global variable get_done with mutex and check it;
            while( get_done != the number of slaves)
                wait and check get_done again later and unlock get_done;
            if(i = the odd number of time step)
                send data horizontally divided;
            else
                send data vertically divided;
        }
        join threads getter;
        destroy mutex;
        close sockets;
    }
}
```

Figure 14. Pseudo Code of Thread *Sender*

3.3.2 Thread *Getter*

Every slave has its own thread *getter* on the master side. The threads *getter* receive the results from slaves line by line and as soon as those data are received by threads *getter*, they will be changed to be stored in the different direction in order to get ready for the next

```

getter()
{
    for( i = 0; till the end of simulating time of irrigation;
        increased by one time step){
        receive the begin line and end line numbers of its block
        allocated to the slave;
        if( i = the odd number of time step){
            receive the calculation results from slave line by line,
            as well as store it to the local variables;
            lock the global variable with mutex;
            change the received data to be stored in the different
            direction;
        }
        else {
            performance the same action as above, just have
            different global variable to store data;
        }
        lock get_done with mutex;
        get_done = get_done + 1;
        if( get_done = the number of slaves and is the given
            interval time)
            change the data to be coordinate values of iso-water
            content curves and send it to draw;
        if( i < the end of calculation time of irrigation){
            signal thread sender to check get_done;
            unlock get_done;
        }
    }
}

```

Figure 15. The part of Pseudo Code of Thread *Getter*

distribution computation on the different direction. At the given interval time, the *getter* will rewrite the data to be coordinate values of iso-water content curves and send it to Spider server to draw dynamic 2D diagram. The part of pseudo code of thread *getter* is show in Fig.3.5.

3.3.3 Thread Receiver

Thread *receiver* is one of three threads running on the slave side and is a *main()* function in *WaterMovement_slv.C*.

Thread *receiver* activates thread *calculator* running on the same slave as soon as it has received the data enough to calculate two lines for *calculator*.

This program declares a continuous memory space as global variable to buffer received data, whose size is same as considered space grid, as well as a point *receiver_pointer* to point where the data have been received on that memory. Another similar continuous memory space is opened as global variable to buffer the calculated results and another *calculator_pointer* to point where the data have been calculated on this memory. If the *calculator_pointer* equals to *receiver_pointer*, that means thread *calculator*

```
Main()
{
    call start_OSB() function to build connection with master;
    create thread calculator and returner;
    for( i = 0; till the end of simulating time of irrigation;
        increased by one time step){
        receive the begin line and end line numbers of the block
        allocated to this slave;
        for(j = the begin line number; j <= the end line number;
            j++){
            receive data from master;
            lock receiver_pointer and calculator_pointer with
            mutex;
            increase receiver_pointer by 2;
            signals thread calculator to check if the
            receiver_pointer equals calculator_pointer;
            unlock receiver_pointer and calculator_pointer;
        }
    }
}
```

Figure 16. Pseudo Code of Thread Receiver

has calculate all the data that thread *receiver* received and it waits more data from *receiver* to calculate. So, as soon as the *receiver* has received other new data, it will signal the calculator to continue its calculation.

```

calculator()
{
    for( i = 0; till the end of simulating time of irrigation;
        increased by one time step){
        lock calculator_pointer and receiver_pointer with mutex to
        check if data available for calculation;
        while( data are not available)
            wait the signal from thread receiver and unlock
            calculator_pointer and receiver_pointer;
        while(1){
            lock calculator_pointer and receiver_pointer;
            if(this slave has finished to calculate all the block
                allocated)
                get out of the while(1) loop and go to the
                calculation of next new time step;
            while( there are not available data to be used to
                calculate and this slave has not done with the
                whole block allocated)
                wait till thread receiver signals it data
                available;
            unlock receiver_pointer and calculator_pointer;
            calculate two lines of water contents;
            lock the global variables calculator_pointer,
            receiver_pointer, ...;
            calculator_pointer is increased by 2;
            if( calculator_pointer is greater than the end line of
                the block allocated)
                slave has completed computation of the whole block
                allocated;
            signal thread returner to return the previous calculated
            results to master;
            unlock the global variables calculator_pointer,
            receiver_pointer, ...;

        }
    }
}

```

Figure 17. Pseudo Code of Thread Calculator

3.3.4 Thread Calculator

Thread *calculator* is responsible for calculating the water contents. Every time when it finish two line computation, it will go back to check if its *calculator_pointer* equals *receiver_pointer*, if so, it will wait till thread *receiver* signal it that other new data have been received, otherwise, if *receiver_pointer* is greater than *calculator_pointer*, which means there are more data not calculated, thread *calculator* will continue to calculate them.

Just before thread *calculator* goes to calculate the next two lines, it will signal thread *returner* to return the previous calculated results to master. There is another pointer *returner_pointer* to point where the data has been returned on the memory opened for storing the calculated results as mentioned in section 3.4.3.

3.3.5 Thread Returner

Thread *returner* returns the calculated results line by line. The continuous memory storing the calculated results has two pointers *calculator_pointer* and *returner_pointer*. If the values of these two pointers are equal, that means there are not available results to be returned to the master or all the available results have been returned.

```

returner()
{
    for( i = 0; till the end of simulating time of irrigation;
        increased by one time step){
        lock the global variables calculator_pointer and
        returner_pointer;
        while( there are not available data to be returned)
            wait the signal from thread calculator and unlock
            calculator_pointer and returner_pointer;
        return the begin line and end line numbers of the block
        allocated to this slave.
        While(1){
            lock calculator_pointer and returner_pointer;
            if( this slave has finished the calculation of the whole
                block)
                get out of while(1) loop and go to the next new time
                step;
            while( there are not available data to be calculated and
                slave has not done with returning the whole
                block)
                wait till thread calculator signal data available
                and unlock calculator_pointer and
                returner_pointer;
            return one line of data to master with line number;
            lock returner_pointer, ...;
            returner_pointer increased by 1;
            if( returner_pointer is greater than the end line number
                of the block allocated it)
                this slave has completed to return the whole block;
            unlock calculator_pointer, ...;
        }
    }
}

```

Figure 18. Pseudo Code of Thread Returner

CHAPTER FOUR

SOFTWARE TESTING AND PERFORMANCE ANALYSIS

In this section we evaluate the performance of the Spider II system when executing SWaMM application program. We compared the performance of the distributed program running on the Spider II system with a sequential program running on a single PC in the Department of Computer Science, CSUSB. We also compared the performance with a sequential program running on a single PC at the Department of Environment, UCR. The analysis of the performance is discussed in this section as well.

4.1 Testing Environments

In the Department of Computer Science, CSUSB, all the measurements were performed on 1.8 GHz AMD Athlon systems running Red Hat Linux 7.3 with 256 MB of memory, connected by a 100Mb/sec Ethernet local network. The gcc 2.96 is the compiler that we used to compile the Spider II system and SWaMM software.

In the Department of Computer Science, UCR, the SWaMM sequential program is tested on a single computer with Intel Pentium III of CPU (500 MHz) and 256 MB of memory.

The Microsoft visual C++ 6.0 is used to compile the sequential program.

4.2 Testing Methods

We implemented a distributed algorithm of SWaMM described in section 3.1 in Spider II system and compared it with the sequential version of the program. To test the performance, we run the SWaMM software with different sizes of grid as show in the Figure 3 and record their execution time separately. The testing sizes of the grid range from (200×200) to (1000×1000) . For various size grids, the initial conditions input by the users are assumed to be same as the following:

Saturated Water Content ($\frac{cm^3}{cm^3}$): 0.45

Water Source ($\frac{cm^3}{cm^3}$): 0.35

Initial Water Content ($\frac{cm^3}{cm^3}$): 0.2

Error Tolerance: 0.1

Time Step (Minutes): 0.1

Horizontal Distance Step (cm): 2

Vertical Distance Step (cm): 2

Irrigation Time (Minutes): 5

So the grid size, for example (200×200) , is calculated from the horizontal distance 400 cm and vertical distance 400 cm required by users at beginning of the program if the distance steps on both directions are 2 cm, that is, from $400 / 2 = 200$ cm in the horizontal direction and $400 / 2 = 200$ cm in the vertical direction.

Based on the distributed algorithm of an iterative calculation as described in Section 3.1, the irrigation time is tested by 5 minutes. So the calculation is iterated up to 50 times if the time step is 0.1 minutes, which is sufficient to prove the quality and reliability of the SWaMM software running on Spider II.

The type of the tested soil is a homogeneous light loam whose water movement parameters are $D(\theta) = 278.3 (\theta/\theta_s)^{8.05}$ (cm^2/min) for Soil Water Diffusivity and $K(\theta) = 1.42 (\theta/\theta_s)^{10.24}$ (cm/min) for Hydraulic Conductivity.

The water source is buried at 35 cm below soil surface on the left side of soil profile.

The boundary conditions except around the water source point are assumed to be kept as the initial water content θ_a with time going after irrigation. The boundary condition around the water source point is constant.

4.3 Testing Results

4.3.1 Computation Time

Table 4.1 shows the results of execution times of grid sizes (200×200) to (1000×1000) using 1 to 10 machines. Figure 19 illustrates the graphs of the execution time for grid sizes (200×200) to (1000×1000) using 1, 3, 5, 7, 9 machines at CSUSB.

First, for the sequential program run on 1 machine, the execution time at UCR is about 4 times that at CSUSB since the clock rate of machines at CSUSB is much higher than that at UCR, that is, the CPU of the computers at CSUSB is 3.6 times as fast as the CPU at UCR.

Second, Figure 19 shows that all the results from distributed program are faster than the sequential program at CSUSB regardless of grid size. Comparing with the sequential program, as the grid size gradually increases from (200×200) to (1000×1000) , the computation times are significantly decreased no matter how the computation tasks are distributed on 3, 5, 7 or 9 machines. But when comparing the execution times between 7PC and 9PC for ever grid size, we find that the difference of execution times

Size Machine	200 x 200	300 x 300	400 x 400	500 x 500	600 x 600	700 x 700	800 x 800	900 x 900	1000 x 1000
1	ucr 83	283	641	1236	2074	3261	4848	6814	9291
	csusb 20	71	159	330	482	750	1111	1528	2123
3	13	27	53	102	148	224	336	463	624
4	12	30	43	91	127	172	256	293	474
5	11	24	35	75	102	147	212	354	392
6	10	21	30	75	86	129	184	251	241
7	10	19	30	62	77	112	164	226	303
8	10	16	30	54	70	123	170	218	358
9	11	18	30	49	67	100	149	203	264
10	12	17	28	45	61	94	144	188	227

Table 1. Comparison of Execution Time

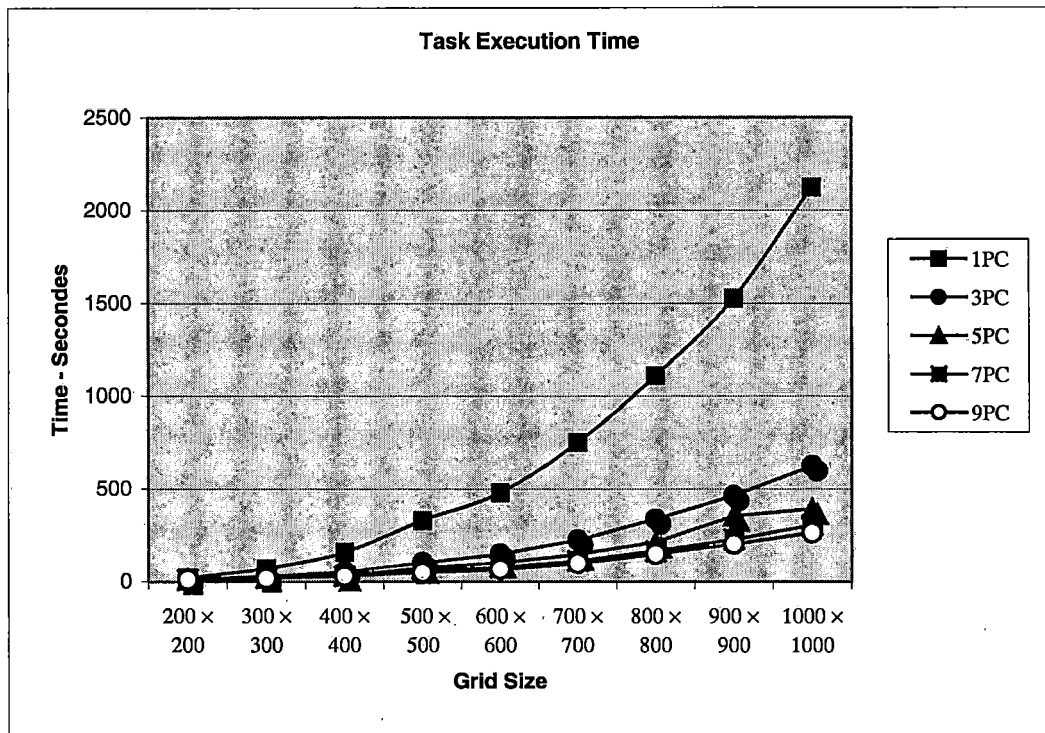


Figure 19. Comparison of Execution Time

between them is smaller than the difference between 3PC and 5PC. This is reasonable if we consider increasing communication times as the number of nodes is increased. This demonstrates that communication time will affect the computation time when the number of nodes has reached a threshold.

4.3.2 Speedup and Efficiency

The speedup and efficiency results are shown in Table 4.2, where Speedup and Efficiency are defined as the

Size Mac	200 x 200		300 x 300		400 x 400		500 x 500		600 x 600		700 x 700		800 x 800		900 x 900		1000 x 1000	
	SP	EF	SP	EF	SP	EF	SP	EF	SP	EF	SP	EF	SP	EF	SP	EF	SP	EF
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3	1.5	0.5	2.6	0.9	3.0	1.0	3.2	1.1	3.2	1.1	3.3	1.1	3.3	1.1	3.3	1.1	3.4	1.1
4	1.7	0.4	2.3	0.6	3.7	0.9	3.6	0.9	3.8	1.0	4.4	1.1	4.3	1.1	5.2	1.3	4.5	1.1
5	1.8	0.4	2.9	0.6	4.6	0.9	4.4	0.9	4.7	0.9	5.1	1.0	5.2	0.9	4.3	0.9	5.4	1.1
6	2.0	0.3	3.3	0.6	4.6	0.8	4.4	0.7	5.6	0.9	5.8	1.0	6.0	1.0	6.1	1.0	8.8	1.5
7	2.0	0.3	3.7	0.5	5.3	0.8	5.3	0.8	6.2	0.9	6.7	1.0	6.8	1.0	6.8	1.0	7.0	1.0
8	2.0	0.3	4.4	0.6	5.3	0.7	6.1	0.8	6.9	0.9	6.1	0.8	6.5	0.8	7.0	0.8	5.9	0.7
9	1.8	0.2	3.9	0.4	5.3	0.6	6.7	0.7	7.2	0.8	7.5	0.8	7.4	0.8	7.5	0.8	8.0	0.9
10	1.7	0.2	4.1	0.4	5.7	0.6	7.3	0.7	7.9	0.8	8.0	0.8	7.7	0.8	8.1	0.8	9.3	0.9

Table 2. Comparison of Performance

Note: (1) SP is SpeedUp
(2) EF is Efficiency

following,

$$\text{Speedup} = \frac{\text{The Execution Time of Sequential Program}}{\text{The Execution Time of Distributed Program}}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{The Number of Nodes}}$$

The speedup and efficiency for 3, 5, 7, 9 machines are illustrated in Figure 20 and Figure 21. First, the speedups go up as the number of nodes is increased at every

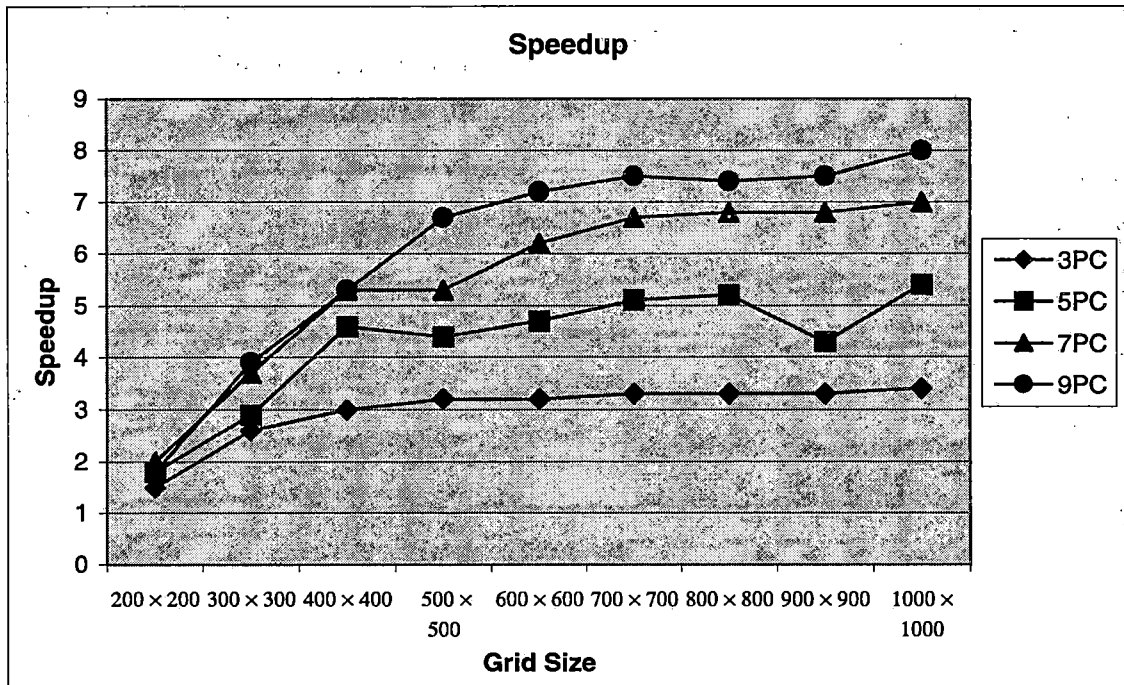


Figure 20. Comparison of Speedup

grid size. Second, at the small grid size of (200 x 200), the approximately equal speedup is achieved for the various

numbers of machines. When the grid size goes up to (300 × 300) and (400 × 400), the speedups are significantly increased for every number of machines and the efficiencies for these two grid sizes are also increased fast as showed in Figure 21. But after running size (400 × 400), the speedups and efficiencies for various numbers of machines tend to level out as the grid size increases. That means

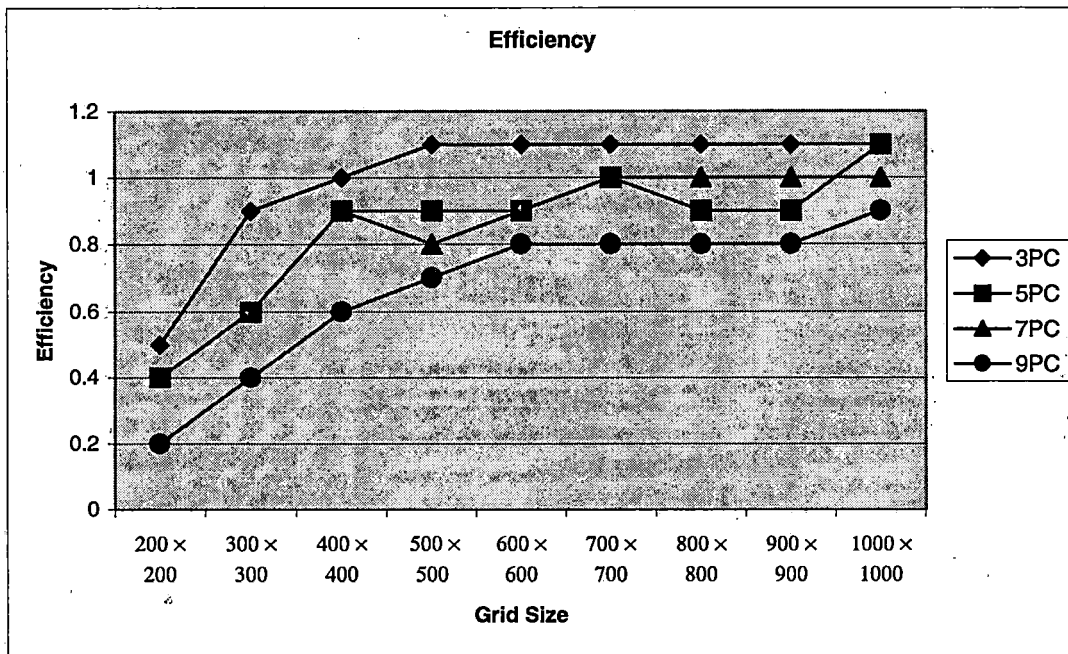


Figure 21. Comparison of Efficiency

the capacity of the underlying computer system is almost achieved to the maximum when the grid size is reached to

(500×500) or (600×600). Third, as what we can see from Figure 21, the efficiency remains higher with all grid sizes when using 3 machines than using 9 machines. This proves there is a great influence of communication time upon the computation time.

CHAPTER FIVE

MAINTENANCE MANUAL

Maintenance is important to keep the software up and running. Since the Spider project is an on-going project, maintenance manual also helps its development in future.

5.1 System Requirements

The distributed program of SWaMM has been tested on RedHat Linux7.0 and Linux7.3. The sequential program has been tested on WindowXP beside RedHat Linux7.0 and Linux7.3. We believe they should take no extra efforts to be installed and run on other platform. SWaMM also requires g++ compiler to compile it. For the diagram of SWaMM which is implemented in Java, the Java 1.2 or up needs to be installed.

5.2 Obtaining a Copy

You can get a copy of SWaMM software and Spider II source code by ftp://spider.ias.cusb.edu/pub/SWaMM_SpiderII.tar.gz, which contains all the necessary c/c++ classes for Spider II and SWaMM software and Java classes for SWaMM diagram and configuration files. The documentation is also included in

this file. After you downloaded the file, you need to extract the tar file. Under a shell prompt, type:

```
tar zxvf SWaMM_SpiderII.tar.gz
```

5.3 Directory Organization

Under the SWaMM_SpiderII directory, there are three subdirectories: bin, java, src and output, two files: ports.conf reghost.conf and a document: readme.

bin: This directory contains executable files for installation of Spider II and daemon startup for Java diagram of SWaMM.

java: This directory contains two subdirectories: classes and src. The classes subdirectories include all the executable files for startup of SWaMM diagram, which are generated from the java codes in subdirectories src after compilation.

src: This directory contains Spider II and SWaMM source code.

ports.conf: This file is for Spider II to configure the port numbers of all the machines which you are going to access by Spider II system.

reghost.conf: This file is for Spider II to configure which machines you are going to use.

output: This directory contains prime results after running a small application prime search to check if Spider II has succeed to install.

readme: This file explains how to install Spider II system and startup SWaMM software.

5.4 Checking for Thread Support

Because SWaMM software is implemented in multi-

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
    printf("POSIX version is set to %ld\n", _POSIX_VERSION);

    if(_POSIX_VERSION < 199506L){
        if(_POSIX_C_SOURCE >= 199506L)
            printf("Sorry, you system does not support POSIX1003.1c
threads\n");
        else
            printf("Try again with -D_POSIX_C_SOURCE=199506L\n");
    }
    else{
        printf("You system supports POSIX1003.1c threads, \n");
#ifdef _POSIX_THREAD_PRIORITY_SCHEDULING
        printf("including suport for priority scheduling \n");
#else
        printf("but does not support priority scheduling\n");
#endif
    }

    exit(EXIT_SUCCESS);
}
```

Figure 22. Program *thread1.c*

threads, we suggest you to check that your platform actually has support for threads, and is compliant with the POSIX. We can do this check in a very easy way to write a very short program, compile it and see what we get.

Figure 22 shows a program *thread1.c* that doesn't do anything other than test for threads support in the libraries that you use when compiling a program. If your machine support for POSIX threads, the test result should be what is reported in Figure 23. If your setup did not report support, or failed to compile at all, you should try compiling the program, setting `_POSIX_C_SOURCE` on the command line, like this:

```
cc -D_POSIX_C_SOURCE=199506L thread1.c -o thread1
```

If your system still doesn't report support for POSIX1003.1c threads, then you may not be able to run SWaMM.

```
$ ./thread1
POSIX version is set to 199506
Your system supports POSIX1003.1c threads,
Including support for priority scheduling
```

Figure 23. Test Result for Program *thread1.c*

5.5 Installation of Spider II System

In order to install Spider II system, you need to do:

(1) check if you have `.rhosts` files under your home directory. If you have not, just create it as the following example:

```
jb359-0.csci.csusb.edu llwang
```

```
jb359-1.csci.csusb.edu llwang
```

```
jb359-2.csci.csusb.edu llwang
```

```
...
```

and add as many machines as you want in a lab where you are. You need to change "llwang" in the example to your user name. After creating `.rhosts` file, restart your machine.

(2) go to `SWaMM_SpiderII` directory and change the `ports.conf` and `reghost.conf` files to add more machines if you want.

(3) read the `readme` file under `SWaMM_SpiderII` directory. Following the steps in `readme` to configure `Spider_II` system.

a. There is a file named `profile.h` in `src` directory.

You need to change `profile.h` for paths of

"`Spiderpath`", "`DATA_FILE`", "`HOSTFILE`",

"`Task_Flag`", and "`Prime_File`" to your directory.

- b. Change two hostnames for "Task_Server" and "Task_Mirror" to which you like to run your Task Server and Task_Mirror, respectively.
- c. Change port numbers for "T_manager_port", "T_mirror_port", "OSB_port" and "Reg_port" to which you want to use and they should not be less than 6000.
- d. After configuring the profile.h, you may start compiling the spider_II using command "make all". All the generated binary files will be in the directory bin.

(4) Run Task Server on the machine that you defined in profile.h file.

- a. Open a new terminal.
- b. Using command "rsh" or "ssh" to access machine Task Server. For example, "rsh jlb359-0" if you defined machine jlb359-0 is the Task Server.
- c. Run the executable file "task" in directory bin. You can see the list of machines and sentence "Task server is available..." on the screen if you succeed to run the task server.

(5) Run Register Server on any machine you like.

- a. Open another terminal.

- b. Using command "rsh" or "ssh" to access machine Register Server. For Example, "rsh jlb359-6" if you like machine jlb359-6 to be Register Server.
- c. Go to directory bin and run the executable file "reg_d". You can see the list of machines and their CPU idle percentage, i.e. "avhosts 0 => jlb359-0 99". This means that the available host's hostname is "jlb359-0" and its CPU idle percentage is 99%. After that, the hostnames and their priorities are showed on the screen. If you succeed to run Register Server, you can see the sentence "Update to Task Server..." at the bottom line.

(6) Use a simple application prime search to test if Spider II has been succeeded to install. You may run executable file "prime" in directory bin on the machine that you are seating on and see the search results from directory output.

5.6 Running Soil Water Movement Model

In order to open the daemon on master machine for presenting the diagram of SWaMM in real time, run the command `appletd` in the directory `bin`. Then go to directory `java/classes` to run `spiderGUI` with `appletviewer`, like that,

```
appletviewer spiderGUI
```

Then input the initial conditions into the dialog window and submit it. After that, the command `watermovement` with initial conditions will be passed to execute by daemon `appletd`. There are three windows that we can view the execution of applications. One is to show which machines have been selected to do the calculation. These machine hostnames are presented in this window. The second window is about monitoring how the calculation is going on with those machines. The third window demonstrates the 2d diagram of `swamm`. The soil profile is represented by a 2-dimensional cartesian coordinate. The x-axis is the horizontal coordinate, and z-axis is the vertical coordinate, which is the soil depth and considered to be positive downward. After the irrigation is applied from a water source point and the water infiltration has been

started, the 2d graphic of soil water contents will be presented as iso-water content curves. As time goes on and

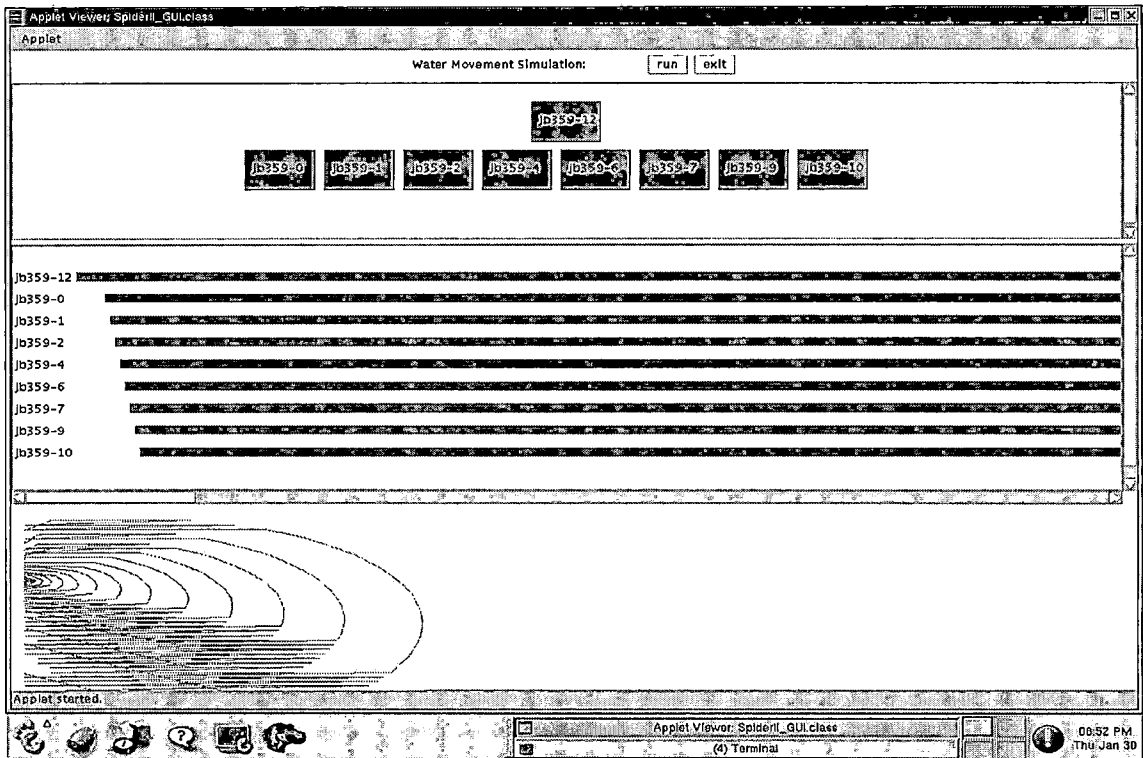


Figure 24. Graph of Soil Water Movement Model

the soil water contents are changed, the iso-water content curves in the 2D graphic will be moved on along the x- and z-axis. The different values of iso-water content curves are represented with the different colors.

5.7 Extension of Soil Water Movement Model

Since SWaMM used the object-oriented program, extending application SWaMM to Spider III system is easy.

5.7.1 Client Program

The client program includes one class Iterative_Calculation and two threads getter and sender. Class Iterative_Calculation provides all the functions required by those two threads.

(1) start()

This function spawns the machines according to the required number by user first. Then it will send initial conditions to them and ask these slaves to start initializing the whole soil profile.

(2) send_data_1() and send_data_2()

These two functions distribute data to the slaves two lines by two lines. Send_data_1() is for sending data at the odd number time, while send_data_2() is going to send data at the even number time.

(3) Change_data_to_draw()

This function calculates the water contents of the whole soil profile to the coordinates which are used to draw the isolines. Because the maximum water content of

soil generally is the saturated soil water content θ_s and the minimum water content is the initial soil water content θ_a . So the calculated water contents at anytime should be between θ_a and θ_s . At the beginning of this function, 40 isolines between θ_a and θ_s are assumed to be considered and values of these isolines are calculated out first. After converting data to coordinate values of the isolines, then send them to daemon on the master machine to draw 2D diagram by function `send_data_to_draw()`.

5.7.2 Server Program

There are one class `WaterMovement_slv` and three threads receiver, calculator and returner in server program. Class `WaterMovement_slv` provides all the functions required by threads.

(1) `start_osb()`

First, this function will create all sockets for communication with master machine. Second, the initial conditions input by user on master machine will be received. Third, it is going to open continuous memory on slave machine according the received initial conditions. Forth, start calculation to initialize the whole soil profile.

(2) `get_begin_end()`

This function is for receiving the begin and end numbers of block assigned by master machine. If you will not change these two numbers during all calculation, you can ignore this function.

(3) `receive_data_1_x()`, `receive_data_1_z()`...

This set of functions is for thread receiver to receive data at an even time or an odd time. Since the distributed algorithm is to send and receive data two lines by two lines, these functions consider of four cases for begin and end numbers of assigned block: begin number is odd value and end number is odd value, begin is odd and end is even, begin is even and end is odd, and begin is even and end is even. These four cases are considered in the functions created for thread calculator as well, for example, `calculator_data_1`, `calculator_data_2` and `calculator_data_3`.

(3) `calculation_start()`

This function starts the calculation of water contents according the received data. All the functions called by `calculation_start()` are related to solve the differential equation in Soil Water Movement Model. Since solving

differential model is very complicate, we suggest you don't change solution method from these functions when you extend SWaMM to Spider III system.

CHAPTER SIX
CONCLUSIONS AND FUTURE
DIRECTIONS

6.1 Conclusions

The Simulation of Soil Water Movement Model on Spider II system project accomplished the following:

- First, this project solves the Soil Water Movement Model that includes a partial differential equation and some auxiliary conditions with the alternating-directions implicit (ADI) difference method. Then according to the data independences of the sequential algorithm ADI, this project develops a parallel and distributed algorithm for distributed calculation of Soil Water Movement Model. In order to hide the communication time during distributed computation, the tasks assigned to every slave are partitioned part by part again, which results that every slave starts calculating as soon as it receives a part of data and starts returning data as soon as the part of data has been calculated. Every slave has actions of

receiving, calculating and returning data at one time point. When the previously received data are being calculated and then returned, the new data are being received. So a parallel algorithm is fulfilled.

- The Soil Water Movement Model is implemented on Spider II distributed system. In order to implement the parallel algorithm, the multi-thread networking programming skill is used with network protocol TCP/IP in C/C++. This multi-thread program uses the mutex to synchronize the accesses to a critical section of code. Using multi-thread and mutex is absolutely essential to improve the computation time because it prevents the interprocess communication between the parent and child when we use fork to create new processes.
- The parallel and distributed algorithm significantly improves the computation time when comparing with its sequential algorithm. The testing size of grid range is from (200×200) to (1000×1000) . The results of the computation time

show that distributed algorithm is faster than sequential algorithm regardless of grid size. The speedups go up as the number of nodes is increased at every grid size. When the grid size goes up from (200×200) to (500×500) , the speedups are increased most significantly for every number of machines comparing with grid size from (500×500) to (1000×1000) . The efficiency remains higher with all grid sizes when using 3 machines than using 9 machines.

- The Java-based user interfaces are implemented as well. In order to monitor the execution of Spider application, there are three windows that can be viewed during execution. One is for viewing which machines have been selected to distributed calculation. Second is to view how the task assigned to every slave is going on. Third window is a dynamic 2D diagram for SWaMM, which the iso-water content curves will be moved on along the x- and z-axis as time goes on and the soil water contents are changed.

6.2 Future Directions

Spider project is an ongoing project. In 2002, Spider II system has been upgraded to Spider III which is a multi-agent based Internet distributed computing system and SWaMM is going to be used by the researchers at the Department of Environmental Sciences, UCR. Below are possible directions that can be extended from the result of this project.

- An extending the real application SWaMM to Spider III is supposed to be a direct and effective measure to test the performance of Spider III system although a very simple application matrix multiplication was programmed on Spider III. The computation time of this application has been significantly improved on Spider II system. The Spider III system has more powerful functions than Spider II system. Applying SWaMM on Spider III system should get better results than on Spider II system.
- The parallel and distributed algorithm in this project is implemented by sending, receiving and calculating data two lines at a time. It is possible that we can design a distributed algorithm

that sends, receives, calculates data one line at a time and compare which performance is better.

- The Soil Water Movement Model solved in this project can have additional parameters that describe a variety of environmental situations based on soil characteristics and nature.

APPENDIX A

GLOSSARY

Agent	A software routine that waits in the background and performs an action when a specified event occurs. For example, agents could transmit a summary file on the first day of the month or monitor incoming data and alert the user when a certain transaction has arrived.
Browser	A program capable of retrieving HTML documents that include references to image and Java bytecode and rendering it into a user-readable document.
Client	The client is a workstation, which requests a service from the Spider system.
Daemon	A UNIX program that runs continuously in the background, until it is activated by a particular event. This word is often used to refer to programs that handle email. The word daemon is Greek for "an attendant power or spirit."

Distributed System	A distributed system is a collection of computers which run their own operating systems or a distributed operating system without having a global memory or a single clock, and computers communicate with each other by exchanging messages over a network.
GUI	Graphical User Interface, an interface that has image as well as words on the screen.
IEEE	Institute of Electrical and Electronics Engineers.
JAVA	A cross-platform programming language from Sun Microsystems that can be used to create animations and interactive features on World Wide Web pages. Java programs are embedded into HTML documents.
OSB	Object service broker (OSB) is the central component of the Spider System to handle the communication between all objects in the system, regardless

of their location, platform or
implementation

Registry Server The server provide the information of
the available servers.

Object-Servers Network computers actually perform the
computation during the distributed
computation.

Socket A communication between two computer
processes on the same machine or
different machines. On a network,
sockets serve as endpoints for
exchanging data between computers.
Each socket has a socket address,
which is a port number plus a network
address.

Soil Water Content	The water lost from the soil upon drying to constant mass at 105 C°; expressed either as the mass of water per unit mass of dry soil or as the volume of water per unit bulk volume of soil.
--------------------	---

Soil Water Movement	Water in soil dynamically flows from higher total water potential to lower water potential, which causes temporal and spatial change of soil water content.
Soil Water Movement Model	It is a technique of reproducing the behavior of the water movement in soil. The prediction and understanding of the water movement processes makes it possible to develop or test various management schemes for controlling the water content of soils.
Spider II	It is a distributed virtual machine running on top of the UNIX or LINUX operating system within the CS network of CSUSB. It features multi-tasking, load balancing and fault tolerance, which optimize the performance and stability of the system.
SWaMM	Simulation of Soil Water Movement Model

Task Manager	To manage the available servers for each task during computation in Spider System.
TCP/IP	The Transmission Control Protocol (TCP) / Internet Protocol (IP). These protocols were developed by DARPA to enable communication between different types of computers and computer networks. The Internet Protocol is a connectionless protocol, which provides packet routing. TCP is connection-oriented and provides reliable communication and multiplexing.
Thread	Threads are sometimes called lightweight processes since a thread is "lighter weight" than a process. All threads within a process share the same global memory. This makes the sharing of information easy between the threads, but along with this simplicity comes the problem of

synchronization.

UML

The Unified Modeling Language (UML) is a third-generation object-oriented modeling language for specifying, visualizing, and documenting the artifacts of an object-oriented system under development.

BIBLIOGRAPHY

- [1] IEEE std 830-1998 IEEE Recommended Practice for Software Requirements Specification
- [2] H.Yuh, *Spider: An Overview of an Object-Oriented Distributed Computing System*. Master Thesis, Department of Computer Science, California State University, San Bernardino, 1997
- [3] Koping Wang, *Spider II: A Component-based Distributed Computing System*. Master Thesis, Department of Computer Science, California State University, San Bernardino, 1997
- [4] Jianhua Ruan, *Spider III: SPIDER III: A Multi-agent Base Internet Distributed Computing System*. Master Thesis, Department of Computer Science, California State University, San Bernardino, 2002
- [5] E.Bresler, B.L.Mcneal, D.L.Carter, *Saline and Sodic Soils----Principles-Dynamics-Modeling*. Springer-Verlag Berlin Heidelberg, New York, 1982, pp. 101-111
- [6] Zhidong Lei, Shixiu Yang, Shenzhuan Xie, *Soil Hydrology*, TsingHua University Press, Beijing, 1988, pp. 264-303
- [7] L. Hluchý, V. D. Tran, L. Halada, et al., *Ground Water*

- Flow Modeling in PVM*, Recent Advances in Parallel Virtual Machines and Message Passing Interface---6th European PVM/MPI Users' Group Meeting, Barcelona, Spain, September, 1999 proceedings, PP. 450-457
- [8] H. M. Deitel, P. J. Deitel, *Java---How to Program*, 3rd edition, Prentice-Hall Inc., New Jersey, 1999
- [9] N. Matthew, R. Stones, "Beginning Linux Programming", 2nd edition, Wrox Press Ltd, Birmingham, UK, November 1999
- [10] W. Richard Stevens, *UNIX Network Programming*, 2nd edition, Prentic-Hall Inc., 1998