OPTIMIZING THE DISTRIBUTED HYDROLOGY SOIL VEGETATION MODEL FOR UNCERTAINTY ASSESSMENT WITH SERIAL, MULTICORE AND DISTRIBUTED ACCELERATIONS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Andrew Adriance

June 2018

COMMITTEE MEMBERSHIP

TITLE:	Optimizing the Distributed Hydrology Soil Vegetation Model For Uncertainty Assessment with Serial, Multicore and Distributed Accelerations
AUTHOR:	Andrew Adriance
DATE SUBMITTED:	June 2018
COMMITTEE CHAIR:	Maria Pantoja, Ph.D. Professor of Computer Sciences
COMMITTEE MEMBER:	Christopher G. Surfleet, Ph.D. Professor of Watershed Management and Hydrolog
COMMITTEE MEMBER:	Chris Lupo, Ph.D. Professor of Computer Science

ABSTRACT

Optimizing the Distributed Hydrology Soil Vegetation Model For Uncertainty Assessment with Serial,
Multicore and Distributed Accelerations

Andrew Adriance

Hydrology is the study of water. Hydrology tracks various attributes of water such as its quality and movement. As a tool Hydrology allows researchers to investigate topics such as the impacts of wildfires, logging, and commercial development. With perfect and complete data collection researchers could answer these questions with complete certainty. However, due to cost and potential sources of error this is impractical. As such researchers rely on simulations.

The Distributed Hydrology Soil Vegetation Model(also referenced to as DHSVM) is a scientific mathematical model to numerically represent watersheds. Hydrology, as with all fields, continues to produce large amounts of data from researchers. As the stores of data increase the scientific models that process them require occasional improvements to better handle processing the masses of information.

This paper investigates DHSVM as a serial C program. The paper implements and analyzes various high performance computing advancements to the original code base. Specifically this paper investigates compiler optimization, implementing parallel computing with OpenMP, and adding distributed computing with OpenMPI. DHSVM was also tuned to run many instances on California Polytechnic State Univisity, San Luis Obispo's high performance computer cluster. These additions to DHSVM help speed-up the results returned to researches, and improves DHSVM's ability to be used with uncertainty analysis methods.

This paper was able to improve the performance of DHSVM 2 times with serial and compiler optimization. In addition to the serial and compiler optimizations this paper found that OpenMP provided a noticeable speed up on hardware, that also scaled as the hardware improved. The pareallel optimization doubled DHSVM's speed again on commodity hardware. Finally it was found that OpenMPI was best used for running multiple instances of DHSVM. All combined this paper was able to improve the performance of DHSVM by 4.4 times per instance, and allow it to run multiple instances on computing clusters.

ACKNOWLEDGMENTS

Thanks to:

- God, for a wonderful creation that allowed for this exploration.
- McKenzie Adriance, for being a wonderful wife and sticking through me these five crazy years of college. Without her I'm sure I would have lost steam around sophomore year.
- Kyle and Marta Adriance, for being wonderful parents who helped guide me to this point in life. Without their love and support I'm sure I wouldn't of made it half as far in life.
- The professors of CalPoly. Who have always looked out for their students and made sure we had the best education experience possible. Nothing makes me prouder to be a mustang than this wonderful community of scholars.
- Pepper Da'Pupper Adriance, whose constant pestering to be taken on walks ensured I wasn't lacking in fresh air or sunlight throughout this fast paced year of graduate school.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CODE LISTINGS

Chapter 1

INTRODUCTION

Hydrology examines our planets most important natural resource, water. The Distributed Hydrology Soil Vegetation Model (henceforth referred to as DHSVM) gives researchers a helpful look at hydrologic processes in watersheds by numerically simulating different elements of the hydrologic cycle based on climate inputs [19]. In DHSVM's current implementation as a serial C program model run times are lengthy. On modern commodity hardware the example data set that comes with the code base takes anywhere from twelve to twenty minutes for a 3 year hydrologic simulation. While that is not an unreasonable amount of time to wait for a single data set, if researchers wish to optimize input parameters and perform uncertainty analysis or evaluate long term processes the programs speed presents a significant bottle neck to researchers. Optimization requires an iterative process of modifying input parameters and re-running DHSVM's simulation. Uncertainty analysis requires numerous model repetitions to produce a sufficient data set to evaluate the modeled environment. These multi-run uses of DHSVM take significant amounts of time. This paper focuses on DHSVM's ability to produce many results sets for uncertainty analysis, but the speed increases can be utilized for other applications such as parameter optimization.

This project is being done in conjunction with the Natural Resources Management (NRES) and Environmental Sciences at California Polytechnic State University of San Luis Obispo. They have a grant from the California Department of Forestry and Fire Protection to study the Caspar Creek watershed with DHSVM. This collaboration was born to assist their research by equipping DHSVM to better utilize the high performance computing resources available to them. Their grant seeks to simulate

different changes to the California Forest Practice Rules on forest roads, silviculture, and water lake protection zones. By creating these simulations researchers will be able to investigate potential hydrologic impacts of these rules.

DHSVM requires detailed climate measurements of solar radiation, relative humidity, wind speed, air temperature, land elevation, roads, and vegetation to create the model. These inputs will be used to run DHSVM within an uncertainty analysis framework. Instead of creating a single optimal parameter set, uncertainty analysis uses a range of parameters through many iterative simulations. This produces a range of acceptable model outputs to allow for improved interpretation of the simulated environment. In its current state DHSVM cannot adequately take advantage of modern hardware to produce results quickly for many runs.

The original collaboration was to configure many instances of DHSVM to run on California Polytechnic State University of San Luis Obispo's computing cluster. However, this paper goes beyond that. In an attempt to make the best use of available computing resources this paper analyzes the original code base to utilize multi-core parallelism on modern hardware, and equip DHSVM to scale for the future.

To the best knowledge of the authors of this paper this is the first attempt at studying the original code base and applying modern high performance computing techniques to it. There have been a number of studies done on the model itself from validation, to parameter optimization [20, 6]. Yao et al. did work on genetic algorithms to optimize DHSVM's input parameters [20]. Their genetic algorithm approach serves as an alternative to the uncertainty analysis method this paper seeks to bolster. The genetic algorithm optimizes a single ideal set of parameters instead of producing a range of results as uncertainty analysis does. DHSVM continuities to be a model of interest, as in 2014 Du et al. did work to investigate DHSVM's effectiveness in a forested mountain watersheds [6]. None of these examples have

focused on making running the model faster. These papers simply seek to analyze and improve upon the results from the model itself.

While DHSVM has not been specifically analyzed there are many ongoing efforts to speed up existing scientific simulations. One such example is the Regional Oceanic Modeling System (ROMS) [4]. While the subject matter of ROMS is different, the computational backbone is similar. ROMS works to traverse 2D and 3D data structure that represent oceanic regions, similar to DHSVM's computation over 2D structures that represent land. This paper will borrow from these similar works for improving DHSVM.

Currently, a single average run of the DHSVM tutorial lasts twelve and a half minutes in this paper's ideal test environment. For a single result set this is not unreasonable. As the number of runs scales up to one thousand it would take almost a week and a half, assuming continuous runs of the calculation without interruption or downtime. Of course, when running real results on compute clusters, other users and potential maintenance makes continuous ideal conditions unlikely to be achieved. It is in this multi-thousand run environment this paper will analyze DHSVM, and work to bring down its run times.

To achieve the desired speed increase this paper will analyze and apply several different techniques. First serial and compiler optimizations are applied to the program. Compiler optimizations will focus on testing both the GNU C Compiler (gcc) and Intel's C Compiler (icc) and their various optimization flags. Serial optimization will focus on improving slow system calls such as printing and memory management. These are especially important as any time saved in repetitively called code will multiply once parallel computing is added to the program. Even if a segment of code isn't repetitively called with-in the normal run of a single simulation, shaving off a few seconds can produce significant savings when aggregated over multiple thousands

of runs. An individual saving of ten seconds will scale to be a savings of almost three hours for every thousand runs.

In addition to the serial optimizations, parallel computing is added to the program. At its core DHSVM is a program that takes a large spatial data structure and loops over it many times to perform calculations. By distributing the work of looping over these spatial data structures to more threads on a computer, performance improves significantly. In addition to measuring the raw performance increase on the test system, this paper also analyzes the scalability of the improvements. As the industry shifts away from clock speed increases in processors and towards adding more cores to computers this becomes increasingly important [8]. Programs will no longer speed up as new processors are released unless they make adequate use of the increasing number of cores. This paper uses OpenMP as its tool of choice for implementing these parallel optimizations.

Finally, distributed computing abilities are added to DHSVM. By adding OpenMPI, DHSVM gains the ability to be running multiple instances of the simulation with varying inputs at any given time. As the goal of this paper is to increase multi-thousand sets of runs of DHSVM using MPI to cooperatively run many instances of the program provides a better overall speed increase oppose to using MPI to speed up a single instance of the program.

This paper finds that DHSVM was able to effectively be optimized for both current commodity hardware, and for scaling up with future hardware. DHSVM was able to perform 4.4 times faster on current commodity hardware and almost 6 times faster on powerful server hardware. This shows that DHSVM is not only able to take better advantage of current hardware, but it will continue to improve as CPUs continue to raise their core counts.

Additionally, DHSVM is now able to use OpenMPI to take advantage of high

performance computing clusters to produce many results. On a four node cluster with each node having 2x 12 core Intel Xeon processors DHSVM can now produce 80 result sets in about the same amount of time it took to produce one result set with the original code on the cluster. The fast production of many result sets will aid researchers interested in uncertainty or optimization applications with DHSVM.

The rest of the paper proceeds as follows: chapter two is the background section. The background section will cover various libraries and concepts utilized in the rest of this paper. Chapter 3 is the Implementation section. Here the exact changes to the program will be discussed. Chapter 4 discusses insights to help future researchers optimize other programs. Chapter 4's insights gained through this project could help save significant time upfront on future projects. Chapter 5 is the validation section. Here the process for validating the integrity of DHSVM and recording program speed improvements are discussed. Chapter 6 and the various changes to DHSVM and how they contributed to improved performance. Chapter 7 will discuss works related to this project that contributed to the foundation of DHSVM's improvements. Chapter 8 summarizes the conclusions of this paper, and finally, Chapter 9 discusses potential future work for DHSVM and high performance computing.

Chapter 2

BACKGROUND

Prior to discussing DHSVM and how to improve it, some background information is required. DHSVM is an interesting piece of software, and it warrants its own introduction as a code base and a tool. Additionally to fully understand the usage of the thousands of results this paper plans to produce uncertainty analysis is described in detail. Finally, a general discussion about types of parallelism and libraries is provided for understanding the implementation details of this paper.

2.1 Distributed Hydrology-Vegetation Model - DHSVM

DHSVM is an open source implementation of Wigmosta et al.'s Distributed Hydrology-Vegetation Model [15, 19]. DHSVM provides an accurate model for vegetation changes, water quality, and run off production for complex terrain. The model takes information about an area of land and climate as input and iteratively and water balances at each time step. Researchers can use this information to investigate the water resources across space and time. DHSVM is specifically used for investigating watersheds, and the water resources they hold.

The source code of DHSVM is implemented in approximately 23,000 lines of C code. While the program originated in the early 1990s with Wigmosta et al. paper, it has been maintained as a collaboration between the Pacific Northwest National Laboratory and the University of Washington [15]. Despite this maintenance, the code base still performs all of its operations serially. There have been studies conducted on the performance of the model, but to the best of the writer's knowledge no one has attempted to apply high performance computing techniques to the code base [10].

DHSVM takes as spatial input a digital elevation model of the land. The model represents vegetation and soil as grids. A water balance calculation is done for each cell of the spatial grid using weather conditions, soil type, typography, and vegetation of the area. DHSVM takes this model constructed by the researchers and iterates through it over a configurable period of time. At the end of this process DHSVM will produce a series of output files that contain information such as total water in the soil and canopy, how much water evaporated over time, and the amount of water gained through precipitation. The dimensions of the model can vary, but are usually in the hundreds. This can produce a data structure with anywhere in the tens to hundreds of thousands of grid cells.

2.1.1 Uncertainty Analysis

While DHSVM is a useful model, there is a level of uncertainty to its outputs. The parameters DHSVM needs to run can be hard to collect, collection methods may be prone to error, and measured values can vary by as much by 150% depending on how and when data is collected [17]. Uncertainty analysis helps combat these levels of variability. It offers an alternative to defining one optimal set of input parameters. Uncertainty analysis is attempting to create equifinality, this recognizes that there is no one optimal set of input parameters, and there may be many valid models that produce equally possible outputs. The generalized likelihood uncertainty estimation (GLUE) procedure is one such analysis method, and is the method that will be used with the final version of DHSVM produced by this paper [3].

Uncertainty analysis can be understood by the general steps researchers take when utilizing it. First, a reasonable range of input parameters must be defined. In the case of DHSVM this allows researchers to express the fact that measured values can vary by 150% without having to settle on one concrete value. Then many instances

of the model must be ran with varying combinations of reasonable input parameters. As outputs are produced, goodness of fit tests determine if a particular set of random parameters and outputs are reasonable. The exact statistical analysis applied will vary depending on use case. Finally, the remaining results can be used to construct a graph that contains a region of reasonable solutions.

2.2 Types Of Parallelism

High performance computing utilizes the multiple cores CPUs to allow for simultaneous calculations to take place. A program may utilize multiple cores by either assigning different pieces of data, or different tasks for each core to process. Each method offers its own distinct advantages. This section will serve as a basic introduction to these types of parallelism.

2.2.1 Data Parallelism

Data based parallelism is the most common form of parallelism. In this style of parallelism different pieces of data are all processed using the same set of code. This form of parallelism can be equated with SIMD(Single instruction, multiple data) operations, or single instruction multiple data operations. Data parallelism comes naturally when many items need to be processed in a similar fashion, such as adding 1 to every element of an array. Many modern CPUs are equipped with SIMD machine instructions and special registers for parallel computations [14]. GPUs also operate with data based parallelism. The nature of computer graphics often has the same operation being applied to every pixel on a screen. As programmers have needed to process more and more data over the years data based parallelism has become a more general purpose item for day to day programming needs [18].

Data parallelism isn't without its flaws. Most data based parallelism paradigms

assume uniform data and uniform operations on that data. If the system allows divergence of operations, such as the use of conditionals, performance can be greatly impacted [7]. This sometimes constrains the types of work loads data based parallelism can effectively handle.

Large matrix multiplications are good examples of data parallelism. A programmer can partition the matrix data by row, column, or square chunks. Each thread can then handle calculating results for its chunk of the matrix. At the end the results of each thread are combined to present a fully multiplied matrix.

2.2.2 Task Parallelism

Task based parallelism focuses on accomplishing more than one task at a time, rather than processing more than one piece of data at a time. This form of parallelism can be equated with MIMD(multiple instruction, multiple data) operations, or multiple instruction multiple data operations. This form of parallelism has different cores operating asynchronously to process different results from different data sets. A good mental model for task based parallelism is how an OS may run more than one process at a time. Each process the OS executes is a 'task' that is being handed to the various cores of a computer. Task parallelism is advantageous for work that operates on irregular data sets that can't be easily generalized into the same set of processing steps. Each task can also implement data based parallelism to provide even greater speedups. Until recently combining data and task based parallelism was difficult [2].

While task based parallelism can be powerful it requires careful consideration by the programmer. The programmer must keep the data being processed by different tasks synchronized. The programmer must also ensure that two tasks can execute independently of each other to avoid incorrect output. Additionally, if tasks are too tightly coupled the time spent synchronizing data can consume any benefit gained by

using multiple cores.

Game engines are a good example of task based parallelism. One thread can be handling the networking code for a multiplayer game, one thread can be handling physics calculations, and another can be handling user input. By having threads working on different tasks it takes less total time to perform logic updates, and thus the game can present new frames to a user faster.

2.3 Parallel Libraries

While languages like C contain built in constructs for parallelism like pthreads, they put much of the parallel programming burden on the developer. There are many libraries that offer useful abstractions to lower the burden of programming. This section will discuss two such popular libraries utilized in this paper, OpenMP and OpenMPI. This section offers a overview of what these libraries have to offer. However, it is not a beginners tutorial for working with them.

2.3.1 OpenMP

OpenMP (Open Multi-Processing) is an industry standard library for shared-memory parallel computing [5]. OpenMP is supported on most systems, and allows for a programmer to easily annotate existing C, C++, and Fortran code to add parallelism. OpenMP excels at data based parallelism by allowing programmers to simply annotate existing loops in a program. Code List 2.1 shows a simple example of an OpenMP program that adds one to each element of an array.

Code Listing 2.1: OpenMP data based loop parallelism

```
//add i to each element of a.  Serial.
for(int i = 0; i < 10; i++) {
    a[i] = a[i] + 1;
}

//add i to each element of a.  Parallel.
#pragma omp parallel for
for(int i = 0; i < 10; i++) {
    a[i] = a[i] + 1;
}
```

OpenMP's shared memory model allows for any thread generated by OpenMP to, by default, access and modify data visible to other threads. The only data that is explicitly safe are variables declared as private in the OpenMP pragmas, or are not accessible due to differences in scope. Due to this shared memory model a programmers largest concern when working with OpenMP are often the race conditions created by two threads trying to work on the same data. OpenMP does provide tools for programmers to avoid and mitigate such race conditions. Firstly, if offers atomic operations to protect the common issues of read and modify operations such as incriminating a variable. Secondly, it offers the ability to define a section of code as critical. Critical sections of OpenMP code will be forced to run serially, with only one thread being able to enter and execute the section of code at a given time.

OpenMP primarily focuses on making data level parallelism easy, but Code List 2.2 shows an example of task based parallelism in OpenMP. While it is possible, it does require more annotation on the side of the programmer.

Code Listing 2.2: OpenMP task based parallelism

```
#pragma omp parallel shared(n,a,b) private(i) {
    #pragma omp sections nowait {
        #pragma omp section
        for (i=0; i<n; i++)
            a[i] = a[i] + 1;
        #pragma omp section
        for (i=0; i<n-1; i++)
            b[i] = b[i] - 1;
    }
}
```

2.3.2 OpenMPI

OpenMPI (Open Message Passing Interface) is a high performance, open source implementation of the Message Passing Interface standard [9]. The Message Passing Interface (MPI) is a standard created and maintained by the MPI Form. MPI's original purpose was to unify the semantics of message passing system, which is a popular paradigm for parallel and distributed computing [9].

MPI revolves around asynchronous workers who explicitly communicate as needed. The explicit passing of data and messages in a MPI system allows additional flexibility over OpenMP. Unlike OpenMP, the workers of MPI tasks don't need to share memory, if needed the messages can be passed over a network. This allows for distributed computing where several computers are all working towards the same goal. The down side of MPI's message passing is that it requires more effort for a programmer to express. Code List 2.3 Shows a simple example of adding one to each elements of an array, contrasted with Code List 2.1 it looks much different than standard C code.

Code Listing 2.3: OpenMPI data parallelism

```
int a[10];
int myA;
int root = 0;
int rank;

//set up MPI environment
MPI_INIT(NULL, NULL);
//Get the id of this worker
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

//worker with rank = 0 sends data to all workers
MPI_Scatter(a, 1, MPI_INT, &myA, 1, MPI_INT, root, MPI_COMM_WORLD);
myA = myA + 1;
//worker with rank = 0 gathers data from all workers
MPI_Gather(&myA, 1, MPI_INT, a, 1, MPI_INT, root, MPI_COMM_WORLD);
//Commnication between workers shut off
MPI_Finalize();
```

While the explicit sending and receiving of information requires addition expression on the part of the programmer, it does make task based parallelism much more natural. In Code List 2.4 two different MPI workers are assigned two different functions to run. If no data needs to be passed between the two processes then nothing else needs to be done. It gives the programmer the benefit of having many independent processes executing, and only sharing information between them as needed.

Code Listing 2.4: OpenMPI task parallelism

```
int root = 0;
int rank;

//set up MPI environment
MPI_INIT(NULL, NULL);
//Get the id of this worker
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if(rank == 0) {
    function1();
} else if (rank == 1) {
    function2();
}

/* ... */

MPI_Finalize();
```

This model of programming offers a distinct advantage, there are no race conditions. Without shared memory there is no way for two MPI workers to trend on one each others work. Thus MPI doesn't offer anything in terms of serialization or atomic operations. Instead MPI focuses on forms of communication. It offers the ability to communicate one-to-one, one-to-all, and all-to-one. Additionally in cases where MPI workers need to all reach the same of execution before continuing the library offers what it calls barriers. MPI barriers are simply a way for a programmer to have every MPI worker stall at a barrier until all workers have reached that point in the code.

Conversely, this model of programming can potentially perform worse. The explicit message passing without shared memory means data synchronization between workers is at the mercy of their communication medium. If the communication medium is a slow ethernet connection MPI's message transfer will adopt the slowness of the ethernet connection. Fortunately, with modern networking hardware this is rarely an issue unless great distances are involved between the two computers.

Chapter 3

IMPLEMENTATION

The goal of this paper is to provide a modern version of DHSVM that can assist future researchers. To accomplish this software upgrades were made in four distinct phases. First additional features were added for users of DHSVM to leverage the new high performance changes and optimize workflows involving uncertainty analysis. Then serial optimizations were added to optimize DHSVM's current code. Thirdly parallel optimization were introduced to take advantage of multi-core hardware. Finally distributed computing was introduced to produce many DHSVM result sets at a time.

3.1 Software Feature Additions

The first added feature allows users to specify the desired number of DHSVM runs. DHSVM will continue to execute new instances of the simulation until the input goal is achieved. Each simulation's output is prefixed with a number for future analysis. The run number is taken as a new runtime argument to the program from the command line. This allows researchers to launch one job for any number of desired simulation results.

```
################################################################################
# CONSTANTS SECTION
################################################################################
[CONSTANTS]                             # Model constants
Ground Roughness      = <0.02-0.05>     # Roughness of soil surface (m)
Snow Roughness        = 0.01            # Roughness of snow surface (m)
Rain Threshold        = -1.0            # Minimum temperature at which rain occurs (C)
Snow Threshold        = 0.5             # Maximum temperature at which snow occurs (C)
Snow Water Capacity   = 0.03            # Snow liquid water holding capacity(fraction)
Reference Height      = 45.0            # Reference height (m)
```

Figure 3.1: An example section of a DHSVM input file with a random range. In this example values for ground roughness will be selected from the range of 0.02 - 0.05m.

In addition to specifying a number of instances the ability to randomize inputs is added. While each instance of the simulation could use a different input file, it would be time consuming for researchers to create that many files. Instead any integer or floating point parameter in an input file can now specify a random range. To specify a number from 0.1 to 0.5 for example a user would list `<0.1-0.5>` as an input parameter. Figure 3.1 shows an example input file with a random range for ground roughness. Each run, in addition to DHSVM's standard output will also output a list of the random values chosen for each input parameter. This allows researchers to analyze and fine tune the random ranges in their input files for models.

The random number range is inclusive. To generate a random number in the range first the delta between the lower bound and upper bound of the range is found. A random double value ranging from 0 to 1 is then produced. The result of the random generation is then multiplied by the delta to find an offset. The offset is then added to the lower bound of the range to finally produce the random number. As the generation allows for the production of 0 and 1 the result ends up being inclusive of the two endpoints.

3.2 Serial Optimization

Prior to spreading work across multiple cores it is important to make the program run faster serially. Any serial speed increases inside of loops will compound once parallel computations are introduced. Serial optimizations additionally allow for performance gains in serial code that can't benefit from additional cores. This paper utilizes both automatic optimizations from compilers, and manual optimization applied to the code.

3.2.1 Compiler Optimization

Modern compilers offer a wide variety of automatic optimization schemes. The general goal of these schemes is to reduce the number of instructions to complete tasks, and improve cache coherency. Reducing instruction counts results in fewer CPU clock cycles being required to complete a chunk of code. Improving cache coherency encourages a program to use data that already exists in a CPU's cache and thus reduces the time spent fetching data. In-depth discussion of these methods such as loop unrolling and data access ordering can be found in other literature such as the survey work of Bacon et al. [1].

Code Listing 3.1: gcc flags

```
gcc -O3
```

Code Listing 3.2: icc flags

```
icc -O3 -prof-use -xCORE-AVX2 -ipo -no-prec-div
```

This paper specifically utilizes the GNU C Compiler (gcc) and the Intel C Compiler (icc). For gcc enabling optimization level 3 and removing all debugging and profiling flags was sufficient for optimization purposes. The same was done for icc. In addition to the basic compilation options icc offers a profile optimization mode. To utilize this option first a version of the program is built with icc that generates profiling information once the program is executed. The statistics created by running the program are then used to recompile the program. This gives the compiler a realistic understanding of memory and instruction usage to better guide the optimization processes. The specific compiler flag set for gcc is shown in Code List 3.1 and the flags for icc are shown in Code List 3.2. The 'ipo' optimization in icc allows the compiler to inline functions that exist in different files. The icc 'no-prec-div' option enables faster floating point divisions. This flag can degrade the accuracy of the floating

point values however. In DHSVM this flag did not significantly impact the results files. Finally in icc the 'xCORE-AVX2' option allows for icc to do processor specific optimizations based on the available instruction set.

3.2.2 System Call Optimization

System calls can be particularly costly. They require a context switch from the requesting process to the kernel, and then for the kernel to execute the code to handle it. This locks up the program and incurs additional clock cycles while the kernel handles the request. Therefore it is desirable to reduce the usage and impact of system calls where possible to avoid the expensive context switches.

The first optimization work is to remove unneeded print statements. By default DHSVM continuously prints out progress makers for the current step of the simulation. The program calls print several times every second to produce these progress makers. For a single running instance of DHSVM these progress markers help users track the simulation to completion. However, in an environment where many simulations will be run at a time constant updates are not vital to the user tracking the programs progress.

The next set of optimizations focus on the memory based system calls such as malloc and free. These calls incur extra overhead as the kernel manages virtual memory. Two different methods are implemented for optimizing these calls. The first optimization uses gperftools as an alternative to the compilers default malloc libraries [12]. While gperftools provides a variety of helpful profiling options, it most importantly contains TCMalloc. It offers up to 4 times the performance over other malloc implementations, and specifically favors threaded environments [13]. TCMalloc gives each thread a cache of memory on top of a central memory cache used for storing larger objects. When objects are freed from TCMalloc they enter a list of available

memory that can be used by future memory requests. These features of TCMalloc all serve to reduce the reliance on context switches into kernel space to handle memory.

In addition to using TCMalloc this paper also investigates manually reducing system memory calls. Each iteration of the simulation mallocs and frees the same data structures. To reduce these system calls malloc and free will only be called for these common data structures once. At the start of the program all these common structures will be initialized with malloc. At the end of the program all these common data structures will be released using free. Each iteration of the simulation where malloc is normally called is replaced with calls to memset for zeroing out the memory of the data structures. The free at the end of each simulation step are no longer needed and are removed all together.

3.3 Parallel Optimization

DHSVM spends a large portion of time traversing the two dimensional grid structure of the spatial data input. It has to traverse it to calculate and aggregate new values each iteration of the program. Additionally there is certain setup up and cleanup work required at each simulation time step that requires traversal of the whole structure. These frequent traversals are the primary target of this paper's parallel optimizations. OpenMP is used for implementation of parallel optimizations.

The traversals appear in the code using nested for loops. One traverses the rows, one traverses the columns. The traversal is parallelized by distributing with fine granularity. Each cell in the two dimensional grid is available to schedule on any thread. This is accomplished by using openMP's collapse feature to turn each iteration of the nested loop into an individual piece of work. The collapse directive effectively rewrites the nested loop at compile time to be a single loop and makes each iteration available to run on any thread. Without the collapse directive a coarser granularity

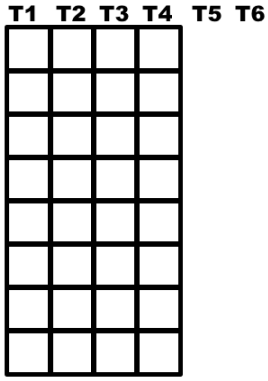


Figure 3.2: A small example of the potential issues of dividing work only by row or column. In this case two threads end up with no work due to a lack of columns in the grid.

can be used to distribute individual rows or columns of the traversal to threads. This coarse granularity is not as scalable, shown in Figure 3.2, and can lead to some threads not getting any work at all. By distributing just rows or columns to threads you end up with either N or M work units. By distributing individual cells you end up with N*M work units. By increasing the number of work units DHSVM will better be able to take advantage of computers in the future with many more cores than currently available.

In addition to these traversals various administrative work for the simulation was distributed to various threads. Each simulation iteration has many malloc and free calls to set up and tear down data structures. All of this work was spread between the available cores. In general any loop with a significant number of iterations (approximately 100 iterations per core) or did non-trivial work (free, malloc, large simulation calculations) were spread to additional cores. Trivial loops (low number of iterations, or simple calculations such as a single add) where left serial due to the administrative costs incurred by distributing work to threads.

For reference appendix B provides the full main file of DHSVM, which contains a significant loop utilizing OpenMP. This can be compared to the original code provided

in appendix A to see an exact application of OpenMP pragmas, and the changes made to facilitate them.

3.4 Distributed Optimization

Distributed computing can be used to speed up the individual run times of a program. This paper seeks to produce many DHSVM results in a short amount of time. As such the optimal strategy is to have each distributed node run a different instance of DHSVM. By having worker nodes run individual instances the amount of communication between nodes is kept to a minimum, allowing for additional results to be produced with reduced overhead. OpenMPI is utilized to handle distributing the various DHSVM instances. Since DHSVM's input file was changed to allow randomized parameters, each DHSVM instance on the cluster can use the same input file with a different set of randomized parameters.

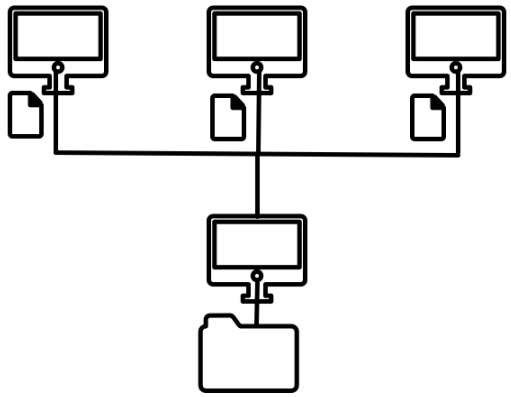


Figure 3.3: Each distributed worker node computes new simulation results. All results are sent to a master node and written to disk.

Each worker node in the cluster runs its own instance of DHSVM. When the simulation finished the results are sent back to the master node and written to disk for future analysis. A visual representation of this setup is shown in Figure 3.3. While it is necessary to aggregate the results to a single location it creates a bottle

neck. DHSVM's runtimes are not highly variable, thus worker nodes may request the master node to write results to disk at the same time. This bottle neck becomes a bigger problem as the size of the cluster increases. To help mitigate this problem the initial DHSVM jobs are slightly offset from one another. The initial variation in start times reduces the chance that two worker nodes will try to have their results written to disk at the same time.

Chapter 4

SUGGESTIONS FOR MODIFYING OTHER PROJECTS

While improving DHSVM this paper found several steps and tools were to assist in the process. These processes and tools would have saved multiple weeks of development time.

4.1 Useful First Steps

In the early stages of modifying the code several steps were developed to help mitigate errors. These two steps covered the root causes of some of the most enigmatic bugs that arose during DHSVM's optimization. After applying these two initial steps to functions before optimization the code worked more reliably.

4.1.1 Non-initialized Memory

One of C's common stumbling blocks is non-initialized memory. Variables without a value assignment end up with whatever random data is in RAM. Before too many optimizations the effects of such issues might not arise. However, once threads start using memory simultaneously chances increase that uninitialized memory will start reading data left behind by other threads.

Compilers will give warnings for uninitialized basic variables, but they won't warn about uninitialized struct fields. These uninitialized fields can cause incorrect output or crash the program. These errors don't give any indication of the uninitialized field, they look just like a coding error. In a code base as large as DHSVM it benefited to side on the error of caution for struct initialization. To avoid the possibility of uninitialized memory in structs all structs where zero initialized in DHSVM using the

syntax in Code List 4.1. This zero initialization syntax will recursively set all field values to zero. This makes numerical values 0 and pointer values NULL. Appendix B provides a real file of code with these modifications applied.

Code Listing 4.1: Safely initialized structs

```
STRUCT_TYPE myStruct = {0};
```

4.1.2 Limiting Scope

When using OpenMP the programmers' main responsibility is analyzing the code to prevent data race conditions. Limiting the scope of variables will help reduce the number of variables in consideration for data race conditions. In older C code variables exist in scope much larger than necessary, often being defined at the top of functions. Any integer not contained within the scope of the OpenMP loop needs to be explicitly made private, or have atomic operations applied to it such as Code List 4.2.

Code Listing 4.2: Explicit private variables

```
int ij;
#pragma omp parallel for private(ij)
for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
        ij = i * j;
        arr[i][j] = ij;
    }
}
```

By pushing variables down as far as possible in scope they become explicitly private such as in Code List 4.3. When working with large legacy code there are often many variables at play in a loop. Reducing the number the programmer has to explicitly work with protecting saves production bandwidth for optimizations. Refer to appendix B to see a real file of code with these modifications applied.

Code Listing 4.3: Implicit private variables

```
#pragma omp parallel for
for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
        int ij = i * j;
        arr[i][j] = ij;
    }
}
```

4.2 Useful Tools

Several tools contributed to increasing productivity through helping track bugs and address run time inefficiencies. These tools consisted of gprof, and the PRUNERS tool collection.

4.2.1 gprof

gprof is a GNU standard tool for profiling programs. Using the tool will give a list of function calls in the program along with how many times they were called, how long the functions ran for, and what percentage of run time they account for. This gives a detailed list of functions that can be optimized using high performance computing

methods. To use gprof a program needs to be compiled through gcc using the '-pg' flag. After that gprof can be used to run the program and gather statistics.

This paper used gprof to track expensive operations in DHSVM. However, optimizations were not always applied directly to the slow functions. In general, once a slow function was found both the function itself was investigated, and the series of calls that lead to that function. In certain cases the slow function had a large series of calculations that could be distributed to different cores. More often though these functions where called hundreds of times inside a for loop traversing a data structure. In these cases it was better to distribute the iterations of the for loop to different threads. Then multiple threads could be executing these expensive calculations along with the additional work inside the for loop.

4.2.2 PRUNERS

A particularly interesting tools set is the "Providing Reproducibility for Uncovering Non-deterministic Errors in Runs on Supercomputers" (PRUNERS) project done by the University of Utah and Lawrence Livermore National Laboratory [16]. The PRUNERS toolset offers a variety of tools for trouble shooting common problems in highly optimized programs. The toolset consists of Archer, FLiT, Ninja, and ReMPI. Archer provides troubleshooting output for race conditions in OpenMP programs. FLiT provides floating point variablilty checking for heterogeneous environments. Ninja provides nosiy network injection for testing MPI programs. ReMPI offers a way to capture the communications of an MPI program and replay them for analysis.

This paper particularly found Archer to be a great tool. When modifying large code bases race conditions can be hidden deep within function calls. Archer helped remove some of the guess work required for tracking these unknown race conditions down.

4.3 printf Rounding Errors

One troublesome error found during development had to do with different platforms implementation of printf. This paper found that on macOS 10.13.3 printf would not make consistent rounding decisions when multiple cores were in use. Figure 4.1 shows an example of printf choosing to round the second decimal place differently between two runs of the same program, even though the third decimal places value was the same. This same issue was not encountered when testing on OpenSuse however.

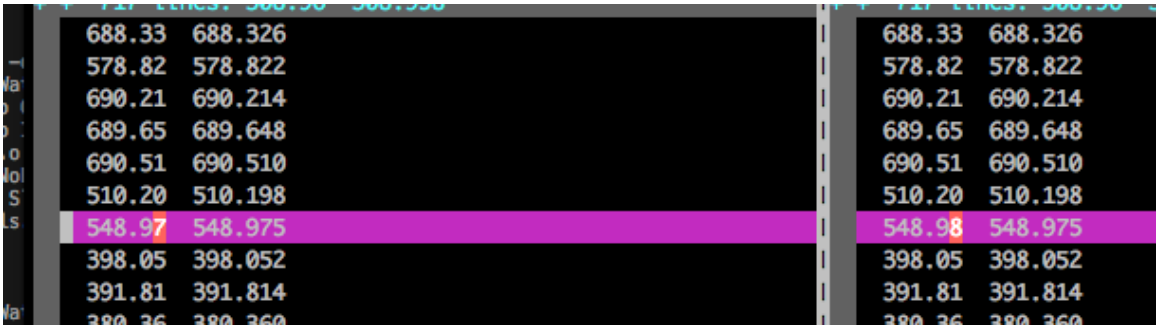


Figure 4.1: Output of %.2f and %.3f from printf

If rounding direction and floating point accuracy are important it can be advantageous to do rounding operations before a printf statement. This way platform specific issues can be avoided, and output can be consistently reliable.

Chapter 5

VALIDATION

Validation of DHSVM was handled as two separate tasks. To ensure correctness every change to DHSVM's code base checked against the output of the original code base. This ensured that program output was not being mangled by optimizations. To ensure the effectiveness of every change to DHSVM's code base time results were gathered each run. To complete these tasks a simple Python test suite was created. This tool automated the process of program correctness checks, and provided timing outputs for gathering results.

Tests were run using various hardware throughout the paper. Three machines were used in particular, their CPU specifications are listed in Table 5.1. For the remainder of the paper Table 5.1 can be referred too. If a graph displays results from a "consumer" CPU then it specifically refers to the Intel Core i7-7700HQ listed in Table 5.1. Additionally appendix C provides the input file used for these experiments.

Table 5.1: CPUs used for experiments

Type	Model	Cores	Clock Speed
Consumer	Intel Core i7-7700HQ	4	2.80GHz
Server	Intel Xeon E5-2695 v3	12	2.30GHz
Research	Intel Xeon Phi 7210	64	1.30GHz

5.1 Guaranteeing Correctness

The Python test suite used the output of the original, known working DHSVM code as an oracle. Each iterative update to the program was then run through the suite, which would compare all new program outputs to the old ones. If any differences

occurred the tool would inform the user which outputs differed and to what degree. Code updates often cause issues later on in data processing rather than immediately where the updates occur. Tracking exactly which outputs differed in the testing suite allowed these downstream issues to be identified by starting with the final result and working backwards towards the change. This gave a simple single path of code to follow. Without this, issue tracking would require starting at a change, and following all code paths affected.

Race conditions and general non-deterministic program behaviors where the biggest concern this paper guarded against. A single vote of correctness from the test suite didn't fully prove the absence of these abnormalities. A program modification would be run at least once in ideal conditions, and once with other tasks on the computer demanding resources to move threads in and out of the CPU. By introducing contention it gave threads a higher chance to swap in and out of the CPU and execute in a different order which can reveal subtle threading errors.

Correctness was not always a binary yes or no when compared to the oracle programs outputs. When compiler options or whole compilers, were switched during the project floating point numbers would not always maintain the same precision. To verify changes to floating point outputs, first a sanity check of the output was completed. This sanity check consisted of ensuring the two outputs still agreed on the most significant decimal places, and that the number itself differed by an insignificant amount (less than 0.01%). After verifying the relative accurateness of the output, the program was run again, but this time compared to its own output. As long as the second programs run matched the first the floating point error was considered insignificant. If they did not match the code was analyzed under the assumption a race condition was introduced.

5.2 Timing Comparisons

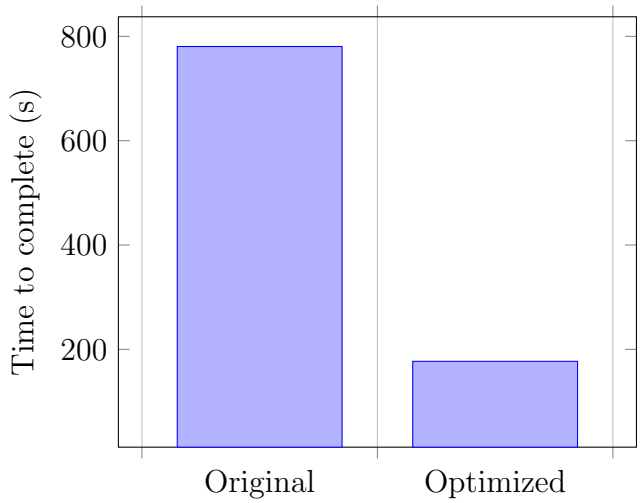
Timing comparisons are gathered as an average of runs in ideal conditions. DHSVM is executed five times, and the average of those runs is used for the final result. Time measurements represent the total wall clock time required for DHSVM to complete from the time the user issues the command to the final output being written.

Chapter 6

ANALYSIS

The original code base of DVSHM took approximately 13 minutes (780.52 seconds) for a single run to complete when running in ideal conditions. After applying all optimizations this paper investigated, a single instance completed in about 3 minutes on consumer hardware and just over 2 minutes on server hardware. This is an overall speed increase for a single run of 440%. Figure 6.1 shows the difference between the original DHSVM code base, and the code base with every optimization enabled. Section 6.1 will analyze the various serial optimizations applied to the code base. Section 6.2 will investigate the parallel optimzations and how well they scaled as cores were added. Section 6.3 will investigate the time and resource trade offs based on how many cores you allocate per DHSVM instance when producing many result sets.

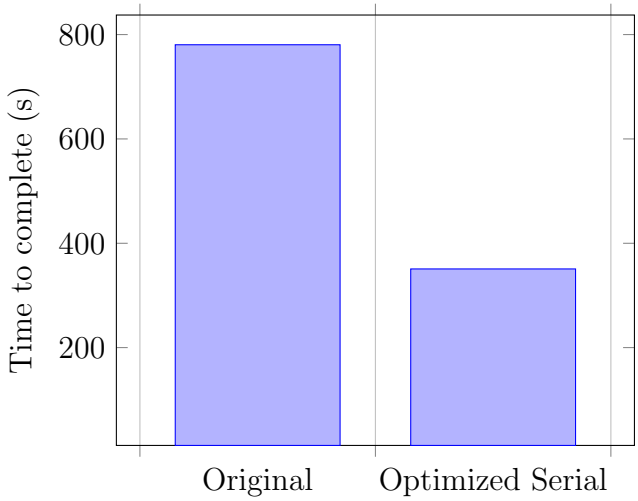
Figure 6.1: A comparison of times between the original program and the most optimized on consumer hardware.



6.1 Serial Optimizations

Serial optimizations accounted for approximately half of DHSVM's speed increases. With serial speed increases the program ran in a little under 6 minutes. That's 220% faster than the original code. Figure 6.2 shows the difference between the original program and the best serially optimized version of the program. The biggest speed increases for serial optimization were gained through compiler optimization and system call optimizations.

Figure 6.2: A comparison of times between the original program and the best serially optimized version on consumer hardware.



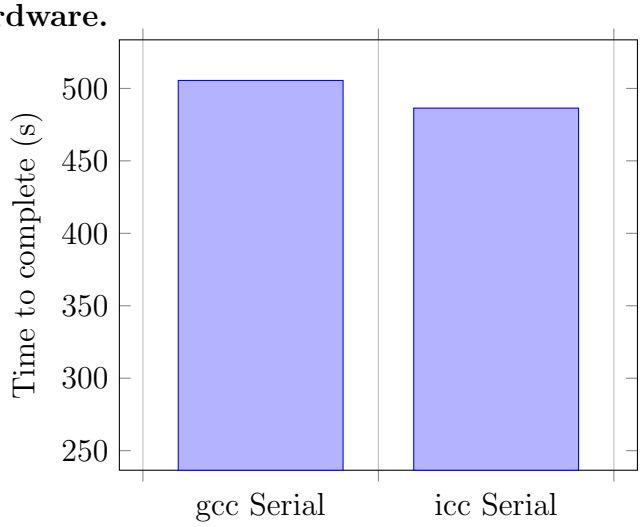
6.1.1 Compiler Optimizations

This paper investigated the various effects of using Intel's specific compiler for Intel CPUs (icc), versus the GNU C compiler (gcc). Figure 6.3 shows the original serial version of the program compiled with both gcc and icc. icc gains a constant 20 second speed increase over gcc. This 20 second constant speed increase also persists when the program is ran with multiple cores. This speed increase is not significant for single runs of DHSVM, but can net large gains over large numbers of runs.

Just using gcc with its basic `-O3` flag shows a significant improvement over the original code base, shaving off almost 300 seconds of run time, for an overall 50% increase in speed. For many uses of DHSVM the additional 20 seconds saved with icc will be unnecessary. However, for the many results set required for uncertainty analysis 20 seconds is a helpful increase in speed.

Using icc does take significantly longer to compile, especially when using profile optimizations. These upfront costs are only incurred once and pale in comparison to the long term time spent producing simulation results.

Figure 6.3: A comparison of times between the original serial program compiled with gcc optimizations and one compiled with icc optimizations on consumer hardware.



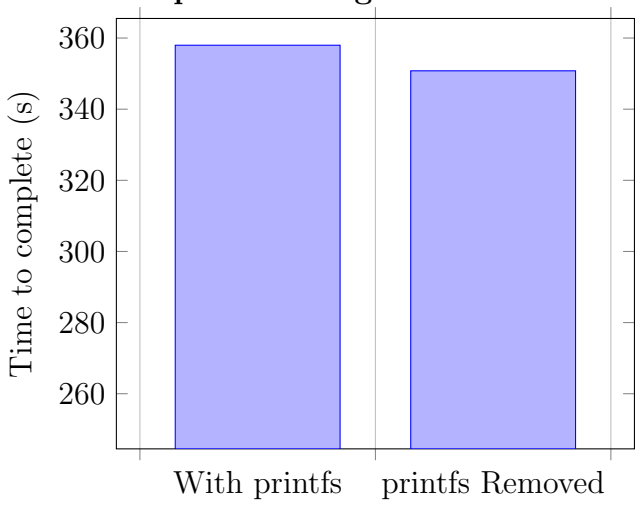
For many large programs compiler optimizations will be an excellent starting point. Enabling optimizations isn't totally free of development time, as it will reveal subtle bugs that may not affect optimized code (such as uninitialized memory). However, these subtle issues will also often cause problems with parallel code and should be worked through before any serious optimization work can take place.

6.1.2 printf Optimizations

The first system call to be optimized was printf. Removing printfs for unnecessary progress updates successfully shaved about 10 seconds off the program. When running many instances of DHSVM a user won't need updates on the progress of an individual run. Instead, the concern is about completed instances of DHSVM. Figure 6.4 shows the most optimized serial version of the program with the printf calls left in, and with the calls removed.

As with icc, this savings is not critical for users who will only run DHSVM a few times, and would likely want to see the progress of a signal simulation instance. A savings of ten seconds translates to almost 17 minutes of time saved for every 100 instances of DHSVM that run in succession. Therefore, the optimization is worthwhile for uncertainty analysis.

Figure 6.4: A comparison of times between the optimized serial program compiled with and without printfs being removed on consumer hardware.

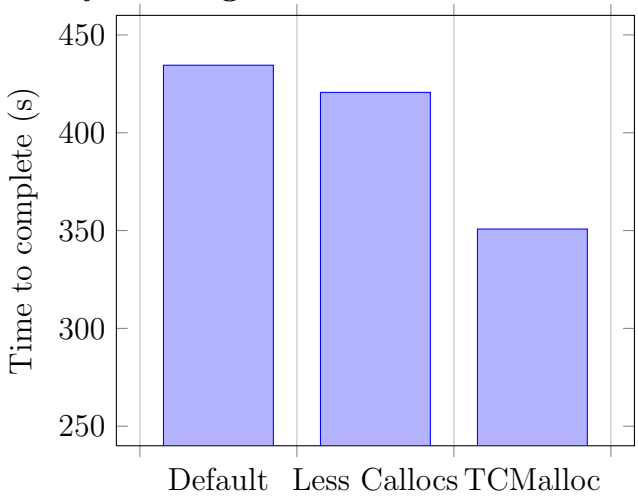


6.1.3 Memory Optimization

Optimizing system calls for memory provided the second best returns for serial optimizations netting an additional 25% performance improvement. Figure 6.5 shows the

difference in run times between using the original code base with the default malloc library, reducing the number of calls to calloc by holding onto memory, and using TCMalloc to replace the default malloc library.

Figure 6.5: A comparison of times between the different memory options. One holds memory and uses memset instead of calloc and free, one uses a 3rd part malloc library. Testing ran with consumer hardware.



Reducing the calls to calloc did give some tangible results, about a 15 second speed increase. However, it required some significant development time to analyze existing code to determine what callocs could be turned into memset calls of existing memory. While 15 seconds is a noticeable increase when running many instances, it was not a good return for the amount of development time invested in it. On the other hand, TCMalloc reduced run times by a minute and a half, a 25% improvement. TCMalloc required very little development time, but does require a version of the library to be available at compile time. TCMallocs optimization for parallel environments and reuse of memory allowed it to excel in DHSVM's environment. Using a third party implementation to improve all uses of the malloc library provided good results for the whole program without having to manually analyze and modify the programs memory usage. In general, replacing existing libraries with use case specific implementations can give significant improvements, as shown by TCMalloc.

6.2 Parallel Optimizations

DHSVM's parallel optimizations reduced runtimes by almost three minutes on consumer hardware. These optimizations took run times from 350 seconds down to 180. Figure 6.6 shows how DHSVM scaled on a commodity Intel CPU with 4 physical cores and Hyper Threading. The graph initially looks like DHSVM can only effectively utilize two cores, however this is not the case. DHSVM's performance actually depends on how many cores are available on the machine physically.

Figure 6.6: A graph that shows the most optimized version of DHSVM's performance relative to number of threads on consumer hardware. The CPU contains 4 physical cores with Hyper Threading.

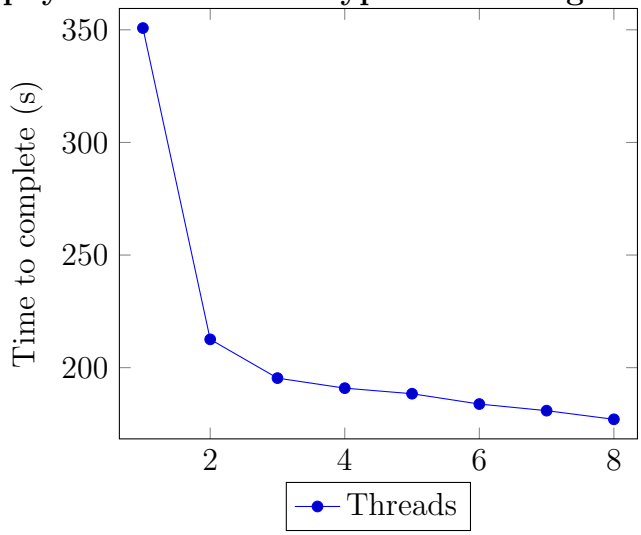
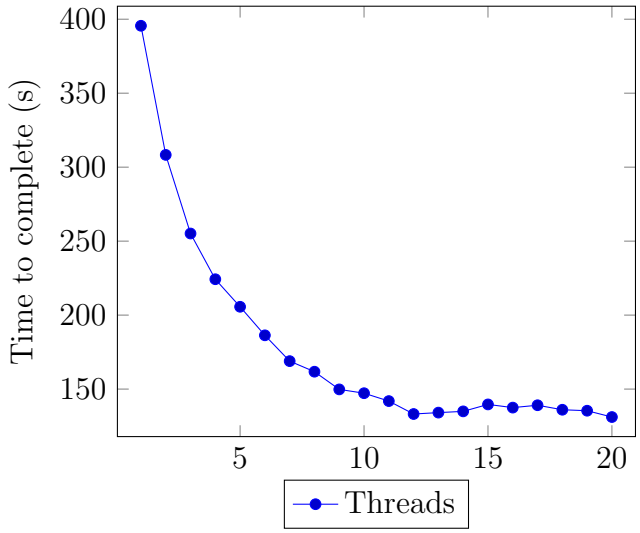


Figure 6.7 shows DHSVM's performance on server hardware with 12 physical core available. This graph gives a clearer picture of DHSVM's performance per thread. First, the two graphs show that virtual cores provided through Hyper Threading on Intel CPUs do not greatly benefit DHSVM. After reaching the physical core limit performance hovers around the same range. Secondly, the graphs show that DHSVM stops getting significant performance improvements after about 75% of the cores are in use. In Figure 6.6 performance improvements become very minimal after 3 cores are in use, and in Figure 6.7 performance improvements start to level off around 9 cores.

Additionally by switching to server hardware DHSVM reduces runtimes by almost a whole minute. The servers individual cores were also slower than the commodity hardware with serial DHSVM taking 395 seconds on the server versus 350 seconds on the commodity CPU. That means with even faster server hardware DHSVM could continue to improve.

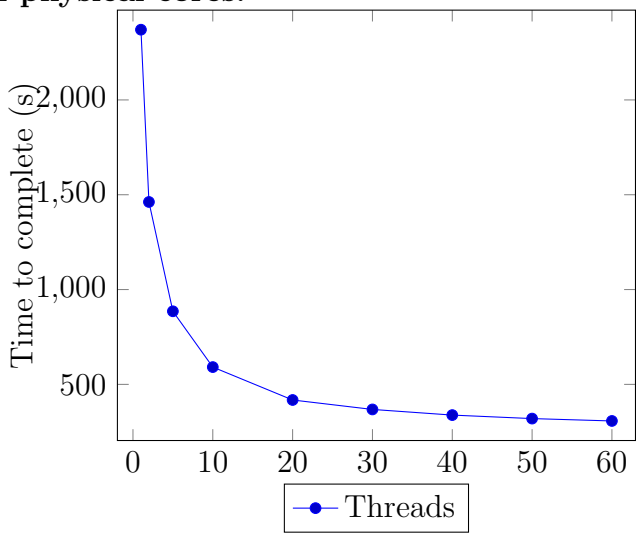
Figure 6.7: A graph that shows the most optimized version of DHSVM's performance relative to number of threads on server hardware. The CPU contains 12 physical cores with Hyper Threading.



DHSVM was able to run twice as fast on commodity hardware with 4 cores, and three times faster on server hardware with 12 cores. This paper was not able to determine why DHSVM's speed decreases when cores are added. It may be due to false cache sharing or scheduling issues. The data show that DHSVM should be able to continue to scale up despite these issues, the net gain per core will simply continue to drop. Exactly how many cores DHSVM can utilize will depend on the size of the land area being simulated.

To test the potential edge of DHSVM's scalability in the future DHSVM was run on a machine with 64 cores. The results in Figure 6.8 show that DHSVM continues to scale in a similar fashion. The curves on all three graphs show that in general DHSVM can now scale in a 1/x fashion relative to the number of cores on the CPU.

Figure 6.8: A graph that shows the most optimized version of DHSVM's performance relative to number of threads on research hardware. The CPU contains 64 physical cores.



The actual times on the research hardware are noticeably slower than the consumer hardware. This is due to the research CPU clock speeds being significantly lower and optimizing for vector operations. Currently, DHSVM needs the higher clock speeds for the complex floating point operations it performs. Additionally DHSVM's code base is not tooled to utilize the vector operations of the CPUs.

Overall these results show that DHSVM can effectively scale to machines with a wide variety of CPU core counts. If in the future massively parallel machines with low powered cores become the norm for computing DHSVM's code base would have to be reworked to efficiently function.

6.3 Multi-core VS. Multi-instance

This papers modifications to DHSVM allow for both multiple cores and multiple instances of DHSVM to be run it is also desirable to understand how to balance computer resources. While allotting more cores per DHSVM instance will return individual result sets faster, the scaling provides diminishing returns. To investigate

this a single server computer was used in three different configurations. The first configuration ran DHSVM serially, allowing for up to 12 simultaneous instances. The second allotted 2 cores to each DHSVM instance, allowing for up to 6 simultaneous instances. The final configuration had 6 cores per instance, allowing for 2 running concurrently. Each set up was asked to produce 1, 6, 12, and 24 sets of DHSVM results. Figure 6.9 shows the raw time recordings from each of these experiments and Figure 6.10 graphs total time divided by the number of result sets produced. The results of this experiment provide three interesting insights.

Figure 6.9: A graph that plots total time required to execute a certain number of DHSVM instances. The graph gives a comparison of time trade offs based on number of cores per DHSVM instance using server hardware.

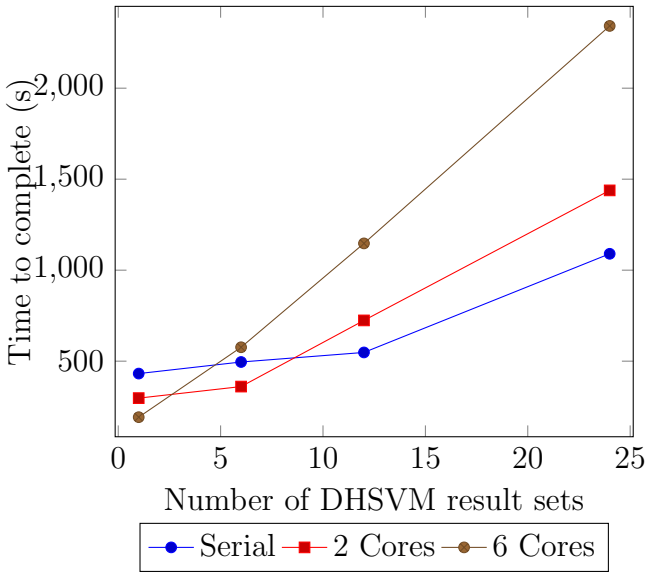
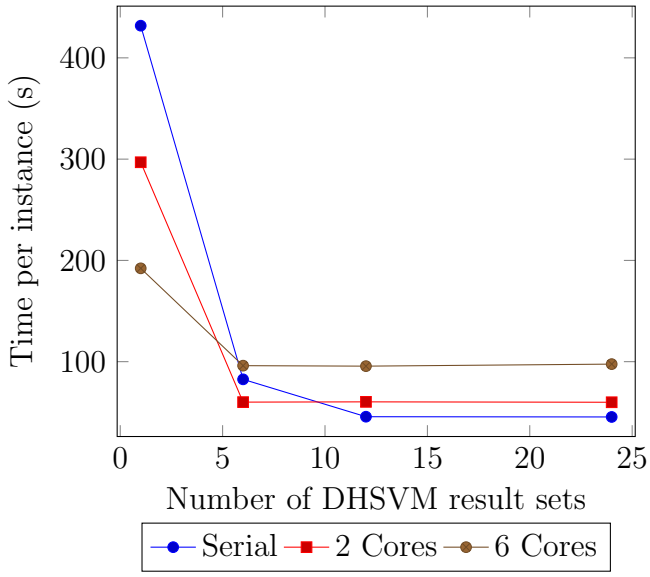


Figure 6.10: A graph that plots time per instance produced. The graph gives a comparison of time trade offs based on number of cores per DHSVM instance using server hardware.



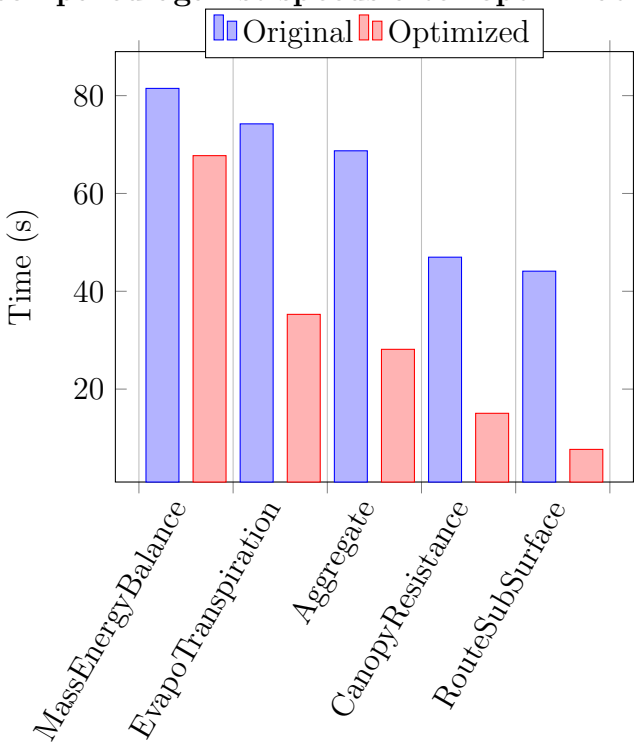
Both Figure 6.9 and 6.10 show that running extra instances of DHSVM scales linearly. This is particularly evident in the data for the instance with 6 cores where additional runs were needed for every instance. Figure 6.10 also shows that once the hardware was fully utilized the time per instance held reasonably constant. These results show that DHSVM's run time required for an arbitrary number of results is reasonably calculable. It simply requires a user to find how long it takes DHSVM to produce its first set of results, and then interpolate linearly to the desired amount of results.

The results gathered also show that DHSVM for uncertainty analysis will scale well to arbitrarily sized compute clusters. Figure 6.9 shows that there is very little overhead to run additional instances of DHSVM. This is best exemplified by the data points of 1, 6, and 12 result sets produced by DHSVM in serial. Therefore, DHSVM will be able to leverage arbitrary numbers of cores and nodes in a compute cluster by running multiple instances of the simulation with minimal overhead.

Most importantly to future users of DHSVM, for large sets of results it makes little sense to use multiple cores for DHSVM. The almost perfect scaling of running multiple instances quickly outperforms the benefits of using multiple cores to accelerate DHSVM. However, the multi-core version is still helpful for researchers who are using workflows other than uncertainty analysis and initial configuration and testing of model inputs.

6.4 Profile Analysis

Figure 6.11: The slowest functions in the original program, and time spent executing them compared against speeds after optimization.



A profiling analysis of run times before and after optimization gives a clear picture of where speedups were gained. Figure 6.11 shows the top five functions DHSVM spends time executing in the original code, and how much time they took after optimizations. The slowest function, `MassEnergyBalance` had some speed-ups from

optimizations, but was hindered from further speedups by data dependencies. Each of the remaining four function operated a significant amount to individual cells of the lands model grid. These operations could easily be split between cores and allowed for noticeable speedups.

Chapter 7

RELATED WORKS

Even without research about accelerating DHSVM it has been an interest to researchers in other ways. Several papers have focused on optimizing the results received from DHSVM. DHSVM has been analyzed for its correctness and applicability in different circumstances. This paper doesn't seek to analyze or improve the correctness of DHSVM as those works have done. Rather, this paper seeks to aid researchers looking to utilize DHSVM by providing a faster feedback loop and a few additional novel features such as randomized input.

DHSVM has many inputs required to run its computational model that can have a significant impact on the output. These inputs are manually gathered by researchers and can have errors and potential variability introduced depending on method and location of data collection. This paper seeks to extend upon these works by making DHSVM capable of using random ranges of input for analysis, along with speeding-up general model execution times to make the overall analysis time faster.

For programmatic optimizations this paper looks outside of DHSVM to other scientific simulations such as ROMS. ROMS, or the Regional Ocean Modeling System, is a scientific model of similar form to DHSVM. While the two subject matters of the simulations are different, their computational skeleton is similar. ROMS already had parallel and distributed computing models implemented into it, but the specific compiler tricks and optimizations used to increase its speed are easily applied to DHSVM as well.

7.1 DHSVM Research

Despite its age DHSVM continues to be analyzed for validation in different environments. In 2014 Du et al. completed a validation and sensitivity test for DHSVM in the Mica Creek Experimental Watershed in northern Idaho [6]. They found that DHSVM did give acceptable output, but required iterative parameter refinement. The refinement of DHSVM's parameters has been an active area of research, and has several recent papers that approach it differently.

In order to improve upon DHSVM's output Yao et al. utilized genetic algorithms to optimize 5 of DHSVM's input variables [20]. Yao et al. were able to show their genetic method was feasible for improving DHSVM output by doing a case study on the Lushi Watershed of the Yellow River Basin. The goal of genetic algorithms is to simulate the process of gene mutation an natural selection in nature. Some set of parameters are randomly mutated over time, and as they are mutated they are measured for fitness. Fitness is represented by the creator of the algorithm as some function that has an expected result to compare against. In this case, fitness would be measured by looking at the outputs of DHSVM itself. This process continues until an acceptable fit is found.

Surfleet et al. used uncertainty analysis to optimize the interpretation of model results rather than just specific input parameters [17]. By varying model inputs within certain ranges and applying the generalized likelihood uncertainty estimation (GLUE) procedure developed by Beven and Binley, a stronger body of results was able to be produced by DHSVM[17, 3].

Both the genetic algorithm and uncertainty analysis require a feedback loop that involves running DHSVM simulations. The genetic algorithm must mutate its input parameters, run a new DHSVM instance, and analyze the fit of the new results. The

uncertainty analysis requires creating many slightly varying result sets to compare and analyze. Uncertainty analysis lends itself better to parallelization as any set of concurrent DHSVM result sets can be generated at a given time. Genetic algorithms must analyze results before creating a new generation of varied parameters, this creates a data dependency with a more serial nature that can limit the parallelization of these tasks. Thus the work of this paper better aids research utilizing uncertainty analysis, although genetic approaches could still benefit from the raw speed improvements of DHSVM.

7.2 ROMS Optimization

The Regional Ocean Modeling System (ROMS) is a scientific simulation that is an open source free-surface, primitive equation ocean model used by the scientific community for a diverse range of applications. ROMS is a highly parallelized simulation that operates over a three dimensional data structure, and comes with the option to utilize either MPI, or OpenMP. The choice to use MPI or OpenMP is made at compile time by the user.

Due to the existence of a modern parallel code for ROMS research has been focused on maximizing its utility. Both Lupo et al. and Bhaskaran & Gaurav optimized ROMS by using hardware specific parallel accelerators [4, 11]. Both utilized the Intel Xeon Phi architecture. Lupo et al. were able to improve the performance of ROMS by 6 times compared to a modern high-performance Xeon CPU without having to change to code base [11]. Bhaskaran & Gaurav were able to speed up ROMS by over 50% on the Xeon Phi. Most interestingly they achieved significant gains just by slightly modifying the compiler time optimization flags [4]. They focused on modifying the code to assist automated compiler vectorization of operations. This involved manual data alignment and simplifying code structures such as nested if statements.

While DHSVM requires the code base to be modernized with parallel features, these articles give insight into maximizing the utility of the parallel features that were implemented. By utilizing powerful compilers, like the Intel compiler used by Bhaskaran & Gaurav, and giving DHSVM access to additional computing hardware this paper was able to push the limits of its additions to DHSVM.

Chapter 8

CONCLUSIONS

This paper successfully updated DHSVM to allow for single instances of DHSVM to better utilize modern hardware and to run multiple instances for producing large sets of results. Applying serial optimizations through both the compilers flags and optimized versions of malloc allowed for DHSVM to half run times serially. Using OpenMP allowed DHSVM to better utilize modern multicore hardware and half run times once again.

Tests against hardware ranging from consumer to research grade show DHSVM's ability to continue to scale as more resources are provided. The scalability follows a 1/x shaped curve relative to the number of cores available. This means DHSVM can effectively take advantage of a wide range of hardware, but the more cores on a machine that are dedicated to a single instance the lower the returns.

OpenMPI allowed for DHSVM to run multiple instances of a simulation to produce many result sets. Analysis of run times required to produce multiple result sets shows that multiple instances of DHSVM can run on a single machine with minimal overhead. This gives DHSVM almost perfect scalability for producing multiple results sets. Researchers using DHSVM can utilize this to run many serial instances of DHSVM on compute clusters to produce many sets of results for uncertainty analysis in a short amount of time.

Overall DHSVM is over 4 times faster than the original code base, with the potential to continue improving with newer hardware in the future. More importantly, the code base is now equipped to allow future researchers to more effectively perform uncertainty analysis.

Chapter 9

FUTURE WORK

There are several ways research on DHSVM as a code base can continue. Firstly, extending off of this work directly, DHSVM's bottleneck as core count rises can be investigated. This would allow for DHSVM to make better use of future hardware, and likely give additional performance gains over the initial findings of this paper. Secondly, OpenMP version 4.5 is on its way to compilers. This new version of OpenMP allows for tasks to be offloaded to accelerators such as GPUs and Intel Xeon Phis. This would allow for additional hardware utilization by DHSVM that would equip it to even better utilize new hardware, and potentially start returning results in real time. Due to the math heavy computations of DHSVM, an optimized math library may also be an additional way to research optimizing DHSVM. Tools such as the BLAS (Basic Linear Algebra Subprograms) library may give noticiable speed benefits in the future.

Instead of doing more research on the existing code base itself, future research can additionally be done on DHSVM as an algorithm. By first expressing DHSVM as an algorithm parallel portions of the code can be identified upfront and optimized for asynchronous computation. This would allow for a new DHSVM code base that is inherently parallel and might scale to future hardware better than the current code base can. It would also serve as an interesting comparison of how much software can be improved if it is rebuilt from the ground up versus being modified in the future.

Another interesting topic of future research would be a survey type paper involving papers that improve and modernize simulations such as DHSVM and ROMs. Such research could serve to determine a generic set of appropriate first steps for improving upon existing code bases. A survey of papers on this topic could aid programmers in

the workforce, and future researchers looking to improve upon existing code bases.

BIBLIOGRAPHY

[1] D. F. Bacon, S. L. Graham, and O. J. Sharp. Compiler transformations for high-performance computing. *ACM Computing Surveys (CSUR)*, 26(4):345–420, 1994.

[2] H. E. Bal and M. Haines. Approaches for integrating task and data parallelism. *IEEE concurrency*, 6(3):74–84, 1998.

[3] K. Beven and A. Binley. The future of distributed models: model calibration and uncertainty prediction. *Hydrological processes*, 6(3):279–298, 1992.

[4] G. Bhaskaran and P. Gaurav. Optimizing performance of roms on intel xeon phi. *Procedia Computer Science*, 51:2854–2858, 2015.

[5] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

[6] E. Du, T. E. Link, J. A. Gravelle, and J. A. Hubbart. Validation and sensitivity test of the distributed hydrology soil-vegetation model (dhsvm) in a forested mountain watershed. *Hydrological processes*, 28(26):6196–6210, 2014.

[7] B. R. Gaster and L. Howes. Can gpgpu programming be liberated from the data-parallel bottleneck? *Computer*, 45(8):42–52, 2012.

[8] D. Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, 2005.

[9] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[10] P. K. Kuraś, Y. Alila, M. Weiler, D. Spittlehouse, and R. Winkler. Internal catchment process simulation in a snow-dominated basin: performance evaluation with spatiotemporally variable runoff generation and groundwater dynamics. *Hydrological Processes*, 25(20):3187–3203, 2011.

[11] P. Lupo and Choboter. Enhancing regional ocean modeling simulation performance with the xeon phi architecture. 2017.

[12] gperftools. gperftools - originally google performance tools. `https://github.com/gperftools/gperftools`. Accessed: 2018-02-22.

[13] gperftools. Tcmalloc : Thread-caching malloc. `https://gperftools.github.io/gperftools/tcmalloc.html`. Accessed: 2018-02-22.

[14] Intel Corporation. Introduction to intel advanced vector extensions. `https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions`. Accessed: 2017-12-11.

[15] Pacific Northwest National Laboratory. Distributed hydrology soil vegetation model. `http://dhsvm.pnnl.gov/`. Accessed: 2017-10-11.

[16] The University Of Utah. Pruners : Providing reproducibility for uncovering non-deterministic errors in runs on supercomputers. `https://pruners.github.io`. Accessed: 2018-02-26.

[17] C. G. Surfleet, A. E. Skaugset, and J. J. McDonnell. Uncertainty assessment of forest road modeling with the distributed hydrology soil vegetation model (dhsvm). *Canadian journal of forest research*, 40(7):1397–1409, 2010.

[18] D. Tarditi, S. Puri, and J. Oglesby. Accelerator: using data parallelism to

program gpus for general-purpose uses. In *ACM SIGARCH Computer Architecture News*, volume 34, pages 325–335. ACM, 2006.

[19] M. S. Wigmosta, L. W. Vail, and D. P. Lettenmaier. A distributed hydrology-vegetation model for complex terrain. *Water resources research*, 30(6):1665–1679, 1994.

[20] C. Yao and Z. Yang. Parameters optimization on dhsvm model based on a genetic algorithm. *Frontiers of Earth Science in China*, 3(3):374–380, 2009.

APPENDICES

Appendix A

ORIGIN MAINDHSVM.C

Code Listing A.1: The original MainDHSVM.c

```
/*
 * SUMMARY:      MainDHSVM.c - Distributed Hydrology-Soil-Vegetation Model
 * USAGE:        DHSVM
 *
 * AUTHOR:       Bart Nijssen
 * ORG:          University of Washington, Department of Civil Engineering
 * E-MAIL:       nijssen@u.washington.edu
 * ORIG-DATE:    Apr-96
 * DESCRIPTION:  Main routine to drive DHSVM, the Distributed
 *               Hydrology-Soil-Vegetation Model
 * DESCRIP-END.cd
 * FUNCTIONS:    main()
 * COMMENTS:
 * $Id: MainDHSVM.c,v 1.42 2006/10/12 20:38:11 nathalie Exp $
 */

/*****************************************************************************/
/*                                  INCLUDES                                 */
/*****************************************************************************/
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "settings.h"
#include "constants.h"
#include "data.h"
#include "DHSVMerror.h"
#include "functions.h"
#include "fileio.h"
#include "getinit.h"
#include "DHSVMChannel.h"
#include "channel.h"

/*****************************************************************************/
/*                              GLOBAL VARIABLES                             */
/*****************************************************************************/

/* global function pointers */
void (*CreateMapFile) (char *FileName, ...);
int (*Read2DMatrix) (char *FileName, void *Matrix, int NumberType, int NY, int NX, int NDataSet,
    ...);
int (*Write2DMatrix) (char *FileName, void *Matrix, int NumberType, int NY, int NX, ...);
```

```
/* global strings */
char *version = "Version 3.1.1";          /* store version string */
char commandline[BUFSIZE + 1] = "";              /* store command line */
char fileext[BUFSIZ + 1] = "";                   /* file extension */
char errorstr[BUFSIZ + 1] = "";                  /* error message */
/*****************************************************************************/
/*                                    MAIN                                   */
/*****************************************************************************/
int main(int argc, char **argv)
{
  float *Hydrograph = NULL;
  float ***MM5Input = NULL;
  float **PrecipLapseMap = NULL;
  float **PrismMap = NULL;
  unsigned char ***ShadowMap = NULL;
  float **SkyViewMap = NULL;
  float ***WindModel = NULL;
  int MaxStreamID, MaxRoadID;
  float SedDiams[NSEDSIZES];      /* Sediment particle diameters (mm) */
  clock_t start, finish1;
  double runtime = 0.0;
  int t = 0;
  float roadarea;
  time_t tloc;
  int flag;
  int i;
  int j;
  int x;                                          /* row counter */
  int y;                                          /* column counter */
  int shade_offset;                               /* a fast way of handling arraay position given
       the number of mm5 input options */
  int NStats;                                     /* Number of meteorological stations */
  uchar ***MetWeights = NULL;    /* 3D array with weights for interpolating meteorological
       variables between the stations */

  int NGraphics;                                  /* number of graphics for X11 */
  int *which_graphics;                    /* which graphics for X11 */
  char buffer[32];

  AGGREGATED Total = {                    /* Total or average value of a  variable over the entire
       basin */
    {0.0, NULL, NULL, NULL, NULL, 0.0},
                                          /* EVAPPIX */
    {0.0, 0.0, 0.0, 0.0, 0.0, NULL, NULL, 0.0, 0, 0.0},
                         /* PRECIPPIX */
    {{0.0, 0.0}, {0.0, 0.0}, {0.0, 0.0}, 0.0, 0.0, 0.0},
                         /* PIXRAD */
    {0.0, 0.0},
                                                                  /* RADCLASSPIX */
    {0.0, 0.0, 0, NULL, NULL, 0.0, 0, 0.0, 0.0, 0.0, 0.0, NULL,
        NULL, NULL, NULL, NULL, NULL, 0.0},
                                                    /* ROADSTRUCT*/
    {0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},          /* SNOWPIX
         */
```

```
  {0, 0.0, NULL, NULL, NULL, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
   0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},                    /*
       SOILPIX */
  { 0.0, 0.0, 0.0, 0.0, 0.0},
                                                /*SEDPIX */
  { 0.0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
                      /*FINEPIX */
  0.0, 0.0, 0.0, 0.0, 0.0, 01, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
};
CHANNEL ChannelData = {NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL};
DUMPSTRUCT Dump;
EVAPPIX **EvapMap = NULL;
INPUTFILES InFiles;
LAYER Soil;
LAYER Veg;
LISTPTR Input = NULL;                   /* Linked list with input strings */
MAPSIZE Map;                                    /* Size and location of model area */
MAPSIZE Radar;                                  /* Size and location of area covered by
    precipitation radar */
MAPSIZE MM5Map;                                 /* Size and location of area covered by MM5 input
    files */
METLOCATION *Stat = NULL;
OPTIONSTRUCT Options;                   /* Structure with information which program options to
    follow */
PIXMET LocalMet;                                /* Meteorological conditions for current pixel */
FINEPIX ***FineMap    = NULL;
PRECIPPIX **PrecipMap = NULL;
RADARPIX **RadarMap   = NULL;
RADCLASSPIX **RadMap  = NULL;
PIXRAD **RadiationMap = NULL;
ROADSTRUCT **Network  = NULL; /* 2D Array with channel information for each pixel */
SNOWPIX **SnowMap             = NULL;
MET_MAP_PIX **MetMap  = NULL;
SNOWTABLE *SnowAlbedo = NULL;
SOILPIX **SoilMap             = NULL;
SEDPIX **SedMap               = NULL;
SOILTABLE *SType          = NULL;
SEDTABLE *SedType             = NULL;
SOLARGEOMETRY SolarGeo;                 /* Geometry of Sun-Earth system (needed for INLINE
    radiation calculations */
TIMESTRUCT Time;
TOPOPIX **TopoMap = NULL;
UNITHYDR **UnitHydrograph = NULL;
UNITHYDRINFO HydrographInfo;  /* Information about unit hydrograph */
VEGPIX **VegMap = NULL;
VEGTABLE *VType = NULL;
WATERBALANCE Mass =                     /* parameter for mass balance calculations */
  { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };

/*****************************************************************************
  Initialization Procedures
```

```
*****************************************************************************/
  if (argc != 2) {
    fprintf(stderr, "\nUsage: %s inputfile\n\n", argv[0]);
    fprintf(stderr, "DHSVM uses two output streams: \n");
    fprintf(stderr, "Standard Out, for the majority of output \n");
    fprintf(stderr, "Standard Error, for the final mass balance \n");
    fprintf(stderr, "\nTo pipe output correctly to files: \n");
    fprintf(stderr, "(cmd > f1) >& f2 \n");
    fprintf(stderr, "where f1 is stdout file and f2 is stderror file\n");
    exit(EXIT_FAILURE);
  }

  sprintf(commandline, "%s %s", argv[0], argv[1]);
  printf("%s \n", commandline);
  fprintf(stderr, "%s \n", commandline);
  strcpy(InFiles.Const, argv[1]);

  printf("\nRunning DHSVM %s\n", version);
  printf("\nSTARTING INITIALIZATION PROCEDURES\n\n");

  /* Start recording time */
  start = clock();

  ReadInitFile(InFiles.Const, &Input);
  InitConstants(Input, &Options, &Map, &SolarGeo, &Time);

  InitFileIO(Options.FileFormat);
  InitTables(Time.NDaySteps, Input, &Options, &SType, &Soil, &VType, &Veg,
             &SnowAlbedo);

  InitTerrainMaps(Input, &Options, &Map, &Soil, &TopoMap, &SoilMap, &VegMap);

  CheckOut(Options.CanopyRadAtt, Veg, Soil, VType, SType, &Map, TopoMap,
           VegMap, SoilMap);

  if (Options.HasNetwork)
    InitChannel(Input, &Map, Time.Dt, &ChannelData, SoilMap, &MaxStreamID, &MaxRoadID, &Options);
  else if (Options.Extent != POINT)
    InitUnitHydrograph(Input, &Map, TopoMap, &UnitHydrograph,
                       &Hydrograph, &HydrographInfo);

  InitNetwork(Map.NY, Map.NX, Map.DX, Map.DY, TopoMap, SoilMap,
              VegMap, VType, &Network, &ChannelData, Veg, &Options);

  InitMetSources(Input, &Options, &Map, Soil.MaxLayers, &Time,
                 &InFiles, &NStats, &Stat, &Radar, &MM5Map);

  /* the following piece of code is for the UW PRISM project */
  /* for real-time verification of SWE at Snotel sites */
  /* Other users, set OPTION.SNOTEL to FALSE, or use TRUE with caution */

  if (Options.Snotel == TRUE && Options.Outside == FALSE) {
    printf
      ("Warning: All met stations locations are being set to the vegetation class GLACIER\n");
    printf
```

```
    ("Warning: This requires that you have such a vegetation class in your vegetation table\n");
  printf("To disable this feature set Snotel_OPTION to FALSE\n");
  for (i = 0; i < NStats; i++) {
    printf("veg type for station %d is %d", i,
           VegMap[Stat[i].Loc.N][Stat[i].Loc.E].Veg);
    for (j = 0; j < Veg.NTypes; j++) {
          if (VType[j].Index == GLACIER) {
            VegMap[Stat[i].Loc.N][Stat[i].Loc.E].Veg = j;
                break;
              }
    }
    if (j == Veg.NTypes) {     /* glacier class not found */
          ReportError("MainDHSVM", 62);
        }
    printf("setting to glacier type (assumed bare class): %d\n", j);
  }
}

InitMetMaps(Time.NDaySteps, &Map, &Radar, &Options, InFiles.WindMapPath,
            InFiles.PrecipLapseFile, &PrecipLapseMap, &PrismMap,
            &ShadowMap, &SkyViewMap, &EvapMap, &PrecipMap,
            &RadarMap, &RadMap, SoilMap, &Soil, VegMap, &Veg, TopoMap,
            &MM5Input, &WindModel);

InitInterpolationWeights(&Map, &Options, TopoMap, &MetWeights, Stat, NStats);

InitDump(Input, &Options, &Map, Soil.MaxLayers, Veg.MaxLayers, Time.Dt,
         TopoMap, &Dump, &NGraphics, &which_graphics);

if (Options.HasNetwork == TRUE) {
  InitChannelDump(&Options, &ChannelData, Dump.Path);
  ReadChannelState(Dump.InitStatePath, &(Time.Start), ChannelData.streams);
}

InitSnowMap(&Map, &SnowMap);
InitAggregated(Veg.MaxLayers, Soil.MaxLayers, &Total);

InitModelState(&(Time.Start), &Map, &Options, PrecipMap, SnowMap, SoilMap,
               Soil, SType, VegMap, Veg, VType, Dump.InitStatePath,
               SnowAlbedo, TopoMap, Network, &HydrographInfo, Hydrograph);

InitNewMonth(&Time, &Options, &Map, TopoMap, PrismMap, ShadowMap,
             RadMap, &InFiles, Veg.NTypes, VType, NStats, Stat,
             Dump.InitStatePath);

InitNewDay(Time.Current.JDay, &SolarGeo);

if (NGraphics > 0) {
  printf("Initialzing X11 display and graphics \n");
  InitXGraphics(argc, argv, Map.NY, Map.NX, NGraphics, &MetMap);
}

shade_offset = FALSE;
if (Options.Shading == TRUE)
  shade_offset = TRUE;
```

```
/* Done with initialization, delete the list with input strings */
DeleteList(Input);

/*****************************************************************************
Sediment Initialization Procedures
*****************************************************************************/
if(Options.Sediment) {
   time (&tloc);
   srand (tloc);
/* Randomize Random Generator */

/* Commenting the line above and uncommenting the line below
   allows for the comparison of scenarios. */
/*  srand48 (0);  */
  printf("\nSTARTING SEDIMENT INITIALIZATION PROCEDURES\n\n");

  ReadInitFile(Options.SedFile, &Input);

  InitParameters(Input, &Options, &Map, &Network, &ChannelData, TopoMap,
                 &Time, SedDiams);

  InitSedimentTables(Time.NDaySteps, Input, &SedType, &SType, &VType, &Soil, &Veg);

  InitFineMaps(Input, &Options, &Map, &Soil, &TopoMap, &SoilMap,
               &FineMap);

  if (Options.HasNetwork){
    printf("Initializing channel sediment\n\n");
    InitChannelSedimentDump(&ChannelData, Dump.Path, Options.ChannelRouting);
    InitChannelSediment(ChannelData.streams, &Total);
    InitChannelSediment(ChannelData.roads, &Total);
  }

  InitSedMap( &Map, &SedMap);

  /* Done with initialization, delete the list with input strings */
  DeleteList(Input);
}

/* setup for mass balance calculations */
Aggregate(&Map, &Options, TopoMap, &Soil, &Veg, VegMap, EvapMap, PrecipMap,
          RadMap, SnowMap, SoilMap, &Total, VType, Network, SedMap, FineMap,
          &ChannelData, &roadarea);

Mass.StartWaterStorage =
  Total.Soil.IExcess + Total.CanopyWater + Total.SoilWater + Total.Snow.Swq +
  Total.Soil.SatFlow;
Mass.OldWaterStorage = Mass.StartWaterStorage;

if (Options.Sediment) {
  Mass.StartChannelSedimentStorage = Total.ChannelSedimentStorage;
  Mass.LastChannelSedimentStorage = Mass.StartChannelSedimentStorage;
}
```

```
  /* computes the number of grid cell contributing to one segment */
  if (Options.StreamTemp)
        Init_segment_ncell(TopoMap, ChannelData.stream_map, Map.NY, Map.NX, ChannelData.streams);

/*****************************************************************************
  Perform Calculations
*****************************************************************************/
  while (Before(&(Time.Current), &(Time.End)) ||
         IsEqualTime(&(Time.Current), &(Time.End))) {
    ResetAggregate(&Soil, &Veg, &Total, &Options);

    if (IsNewMonth(&(Time.Current), Time.Dt))
      InitNewMonth(&Time, &Options, &Map, TopoMap, PrismMap, ShadowMap,
                   RadMap, &InFiles, Veg.NTypes, VType, NStats, Stat,
                   Dump.InitStatePath);

    if (IsNewDay(Time.DayStep)) {
      InitNewDay(Time.Current.JDay, &SolarGeo);
      PrintDate(&(Time.Current), stdout);
      printf("\n");
    }

    /* determine surface erosion and routing scheme */
    SedimentFlag(&Options, &Time);

    InitNewStep(&InFiles, &Map, &Time, Soil.MaxLayers, &Options, NStats, Stat,
                InFiles.RadarFile, &Radar, RadarMap, &SolarGeo, TopoMap, RadMap,
        SoilMap, MM5Input, WindModel, &MM5Map);

    /* initialize channel/road networks for time step */
    if (Options.HasNetwork) {
      channel_step_initialize_network(ChannelData.streams);
      channel_step_initialize_network(ChannelData.roads);
    }

    for (y = 0; y < Map.NY; y++) {
      for (x = 0; x < Map.NX; x++) {
            if (INBASIN(TopoMap[y][x].Mask)) {
                  if (Options.Shading)
                LocalMet =
                MakeLocalMetData(y, x, &Map, Time.DayStep, &Options, NStats,
                                 Stat, MetWeights[y][x], TopoMap[y][x].Dem,
                                 &(RadMap[y][x]), &(PrecipMap[y][x]), &Radar,
                                 RadarMap, PrismMap, &(SnowMap[y][x]),
                                 SnowAlbedo, MM5Input, WindModel, PrecipLapseMap,
                                 &MetMap, NGraphics, Time.Current.Month,
                                 SkyViewMap[y][x], ShadowMap[Time.DayStep][y][x],
                                 SolarGeo.SunMax, SolarGeo.SineSolarAltitude);
                  else
                LocalMet =
                MakeLocalMetData(y, x, &Map, Time.DayStep, &Options, NStats,
                                 Stat, MetWeights[y][x], TopoMap[y][x].Dem,
                                 &(RadMap[y][x]), &(PrecipMap[y][x]), &Radar,
                                 RadarMap, PrismMap, &(SnowMap[y][x]),
                                 SnowAlbedo, MM5Input, WindModel, PrecipLapseMap,
```

```
                                &MetMap, NGraphics, Time.Current.Month, 0.0,
                                0.0, SolarGeo.SunMax,
                                SolarGeo.SineSolarAltitude);

                for (i = 0; i < Soil.MaxLayers; i++) {
              if (Options.HeatFlux == TRUE) {
                if (Options.MM5 == TRUE)
                      SoilMap[y][x].Temp[i] =
                              MM5Input[shade_offset + i + N_MM5_MAPS][y][x];
                        else
                      SoilMap[y][x].Temp[i] = Stat[0].Data.Tsoil[i];
                      }
              else
                SoilMap[y][x].Temp[i] = LocalMet.Tair;
                }

                MassEnergyBalance(&Options, y, x, SolarGeo.SineSolarAltitude, Map.DX, Map.DY,
                          Time.Dt, Options.HeatFlux, Options.CanopyRadAtt, Options.RoadRouting,
                          Options.Infiltration, Veg.MaxLayers, &LocalMet, &(Network[y][x]),
                          &(PrecipMap[y][x]), &(VType[VegMap[y][x].Veg-1]), &(VegMap[y][x]),
                          &(SType[SoilMap[y][x].Soil-1]), &(SoilMap[y][x]), &(SnowMap[y][x]),
                              &(EvapMap[y][x]), &(Total.Rad), &ChannelData, SkyViewMap);

                PrecipMap[y][x].SumPrecip += PrecipMap[y][x].Precip;
              }
          }
    }

        /* Average all RBM inputs over each segment */
        if (Options.StreamTemp) {
          channel_grid_avg(ChannelData.streams);
      if (Options.CanopyShading)
            CalcCanopyShading(ChannelData.streams, &SolarGeo);
        }

#ifndef SNOW_ONLY

    /* set sediment inflows to zero - they are incremented elsewhere */
    if ((Options.HasNetwork) && (Options.Sediment)){
      InitChannelSedInflow(ChannelData.streams);
      InitChannelSedInflow(ChannelData.roads);
        }

    RouteSubSurface(Time.Dt, &Map, TopoMap, VType, VegMap, Network,
                    SType, SoilMap, &ChannelData, &Time, &Options, Dump.Path,
                    SedMap, FineMap, SedType, MaxStreamID, SnowMap);

    if (Options.HasNetwork)
      RouteChannel(&ChannelData, &Time, &Map, TopoMap, SoilMap, &Total,
                   &Options, Network, SType, PrecipMap, SedMap,
                   LocalMet.Tair, LocalMet.Rh, SedDiams);

    /* Sediment Routing in Channel and output to sediment files */
    if ((Options.HasNetwork) && (Options.Sediment)){
      SPrintDate(&(Time.Current), buffer);
```

```
    flag = IsEqualTime(&(Time.Current), &(Time.Start));

    if (Options.ChannelRouting){
        if (ChannelData.roads != NULL) {
          RouteChannelSediment(ChannelData.roads, Time, &Dump, &Total, SedDiams);
          channel_save_sed_outflow_text(buffer, ChannelData.roads,
                                        ChannelData.sedroadout,
                                        ChannelData.sedroadflowout, flag);
            RouteCulvertSediment(&ChannelData, &Map, TopoMap, SedMap,
                                 &Total, SedDiams);
          }
        RouteChannelSediment(ChannelData.streams, Time, &Dump, &Total, SedDiams);
        channel_save_sed_outflow_text(buffer, ChannelData.streams,
                                      ChannelData.sedstreamout,
                                      ChannelData.sedstreamflowout, flag);
      }
    else {
        if (ChannelData.roads != NULL) {
            channel_save_sed_inflow_text(buffer, ChannelData.roads,
            ChannelData.sedroadinflow, SedDiams,flag);
          }
        channel_save_sed_inflow_text(buffer, ChannelData.streams,
            ChannelData.sedstreaminflow, SedDiams,flag);
    }
    SaveChannelSedInflow(ChannelData.roads, &Total);
    SaveChannelSedInflow(ChannelData.streams, &Total);
  }

  if (Options.Extent == BASIN)
    RouteSurface(&Map, &Time, TopoMap, SoilMap, &Options,
                 UnitHydrograph, &HydrographInfo, Hydrograph,
                 &Dump, VegMap, VType, SType, &ChannelData, SedMap,
                 PrecipMap, SedType, LocalMet.Tair, LocalMet.Rh, SedDiams);

#endif

  if (NGraphics > 0)
    draw(&(Time.Current), IsEqualTime(&(Time.Current), &(Time.Start)),
         Time.DayStep, &Map, NGraphics, which_graphics, VType,
         SType, SnowMap, SoilMap, SedMap, FineMap, VegMap, TopoMap, PrecipMap,
         PrismMap, SkyViewMap, ShadowMap, EvapMap, RadMap, MetMap, Network,
         &Options);

  Aggregate(&Map, &Options, TopoMap, &Soil, &Veg, VegMap, EvapMap, PrecipMap,
            RadMap, SnowMap, SoilMap, &Total, VType, Network, SedMap, FineMap,
            &ChannelData, &roadarea);

  MassBalance(&(Time.Current), &(Dump.Balance), &(Dump.SedBalance), &Total,
              &Mass, &Options);

  ExecDump(&Map, &(Time.Current), &(Time.Start), &Options, &Dump, TopoMap,
           EvapMap, RadiationMap, PrecipMap, RadMap, SnowMap, MetMap, VegMap, &Veg,
               SoilMap, SedMap, Network, &ChannelData, FineMap, &Soil, &Total,
               &HydrographInfo,Hydrograph);
```

```
    IncreaseTime(&Time);
        t += 1;
  }

  ExecDump(&Map, &(Time.Current), &(Time.Start), &Options, &Dump, TopoMap,
           EvapMap, RadiationMap, PrecipMap, RadMap, SnowMap, MetMap, VegMap, &Veg, SoilMap,
           SedMap, Network, &ChannelData, FineMap, &Soil, &Total, &HydrographInfo, Hydrograph);

  FinalMassBalance(&(Dump.FinalBalance), &Total, &Mass, &Options, roadarea);

  /*printf("\nSTARTING CLEANUP\n\n");
  cleanup(&Dump, &ChannelData, &Options);*/
  printf("\nEND OF MODEL RUN\n\n");

  /* record the run time at the end of each time loop */
  finish1 = clock ();
  runtime = (finish1-start)/CLOCKS_PER_SEC;
  printf("*******************************************************************************");
  printf("\nRuntime Summary:\n");
  printf("%6.2f hours elapsed for the simulation period of %d hours (%.1f days)\n",
         runtime/3600, t*Time.Dt/3600, (float)t*Time.Dt/3600/24);

  return EXIT_SUCCESS;
}
/*****************************************************************************
  Cleanup
*****************************************************************************/
void cleanup(DUMPSTRUCT *Dump, CHANNEL *ChannelData, OPTIONSTRUCT *Options)
{
        if (Dump->Aggregate.FilePtr != NULL)
          fclose(Dump->Aggregate.FilePtr);
        if (Dump->Balance.FilePtr != NULL)
          fclose(Dump->Balance.FilePtr);
        if (Dump->FinalBalance.FilePtr != NULL)
          fclose(Dump->FinalBalance.FilePtr);
        if (ChannelData->streamflowout != NULL)
          fclose(ChannelData->streamflowout);
        if (ChannelData->streamout != NULL)
          fclose(ChannelData->streamout);
        if (ChannelData->roadflowout != NULL)
          fclose(ChannelData->roadflowout );
        if (ChannelData->roadout != NULL)
          fclose(ChannelData->roadout);

        if (Options->StreamTemp) {
          if (ChannelData->streaminflow != NULL)
                fclose(ChannelData->streaminflow);
          if (ChannelData->streamoutflow != NULL)
        fclose(ChannelData->streamoutflow);
          if (ChannelData->streamISW != NULL)
                fclose(ChannelData->streamISW);
          if (ChannelData->streamNSW != NULL)
        fclose(ChannelData->streamNSW);
          if (ChannelData->streamILW != NULL)
        fclose(ChannelData->streamILW);
```

```
        if (ChannelData->streamNLW!= NULL)
      fclose(ChannelData->streamNLW);
        if (ChannelData->streamVP!= NULL)
            fclose(ChannelData->streamVP);
        if (ChannelData->streamWND!= NULL)
            fclose(ChannelData->streamWND);
        if (ChannelData->streamATP!= NULL)
            fclose(ChannelData->streamATP);
        if (ChannelData->streamBeam != NULL)
            fclose(ChannelData->streamBeam);
        if (ChannelData->streamDiffuse != NULL)
            fclose(ChannelData->streamDiffuse);
        if (ChannelData->streamSkyView != NULL)
            fclose(ChannelData->streamSkyView);
      }
}
```

Appendix B

NEW MAINDHSVM.C

Code Listing B.1: The new MainDHSVM.c

```
/*
 * SUMMARY:      MainDHSVM.c - Distributed Hydrology-Soil-Vegetation Model
 * USAGE:        DHSVM
 *
 * AUTHOR:       Bart Nijssen
 * ORG:          University of Washington, Department of Civil Engineering
 * E-MAIL:       nijssen@u.washington.edu
 * ORIG-DATE:    Apr-96
 * DESCRIPTION:  Main routine to drive DHSVM, the Distributed
 *               Hydrology-Soil-Vegetation Model
 * DESCRIP-END.cd
 * FUNCTIONS:    main()
 * COMMENTS:
 * $Id: MainDHSVM.c,v 1.42 2006/10/12 20:38:11 nathalie Exp $
 */

//#define _USE_MPI_

/*****************************************************************************/
/*                                  INCLUDES                                 */
/*****************************************************************************/
#include "DHSVMChannel.h"
#include "DHSVMerror.h"
#include "channel.h"
#include "constants.h"
#include "data.h"
#include "fileio.h"
#include "functions.h"
#include "getinit.h"
#include "settings.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <omp.h>
#ifdef _USE_MPI_
#include <mpi.h>
#endif

void cleanup(DUMPSTRUCT *Dump, CHANNEL *ChannelData, OPTIONSTRUCT *Options);

/*****************************************************************************/
/*                              GLOBAL VARIABLES                             */
/*****************************************************************************/
```

```
/* global function pointers */
void (*CreateMapFile)(char *FileName, ...);
int (*Read2DMatrix)(char *FileName, void *Matrix, int NumberType, int NY,
                    int NX, int NDataSet, ...);
int (*Write2DMatrix)(char *FileName, void *Matrix, int NumberType, int NY,
                     int NX, ...);

/* global strings */
char *version = "Version 3.1.1";     /* store version string */
char commandline[BUFSIZE + 1] = ""; /* store command line */
char fileext[BUFSIZ + 1] = "";       /* file extension */
char errorstr[BUFSIZ + 1] = "";      /* error message */

#define PRINT 0
#define T_COUNT 4
/*****************************************************************************/
/*                                    MAIN                                   */
/*****************************************************************************/
#ifdef _USE_MPI_
int mpiMain(int argc, char **argv, int id);
int main(int argc, char **argv){
  /* MPI Set Up */
  int comm_sz;
  int my_rank;
  int numRuns;
  if (argc < 3) {
    fprintf(stderr, "\nUsage: %s inputfile numberOfRuns\n\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  numRuns = strtol(argv[2],NULL,0);
  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  //omp_set_num_threads(8);
  //#pragma omp parallel for
  for(int i = my_rank; i < numRuns; i += comm_sz) {
    printf("Number of threads in the current parallel region is %i \n", omp_get_num_threads());
    mpiMain(argc, argv, i);
  }
  MPI_Finalize();
}

int mpiMain(int argc, char **argv, int id) {
#else
int main(int argc, char **argv) {
#endif
  #ifndef _USE_MPI_
  int id = -1;
  #endif
  float *Hydrograph = NULL;
  float ***MM5Input = NULL;
  float **PrecipLapseMap = NULL;
  float **PrismMap = NULL;
  unsigned char ***ShadowMap = NULL;
```

```
float **SkyViewMap = NULL;
float ***WindModel = NULL;
int MaxStreamID = 0, MaxRoadID = 0;
float SedDiams[NSEDSIZES] = {0}; /* Sediment particle diameters (mm) */
clock_t start, finish1;
double runtime = 0.0;
int t = 0;
float roadarea = 0;
time_t tloc;
int flag = 0;
int j = 0;
int shade_offset = 0; /* a fast way of handling arraay position given the number
                      of mm5 input options */
int NStats = 0;        /* Number of meteorological stations */
uchar ***MetWeights = NULL; /* 3D array with weights for interpolating
                               meteorological variables between the stations
                            */

int NGraphics = 0;        /* number of graphics for X11 */
int *which_graphics = NULL; /* which graphics for X11 */
char buffer[32] = {0};
srand48(time(NULL));
struct timespec tstart, tfinish;
double elapsed;
clock_gettime(CLOCK_MONOTONIC, &tstart);

AGGREGATED Total = {
    /* Total or average value of a  variable over the entire basin */
    {0.0, NULL, NULL, NULL, NULL, 0.0},                    /* EVAPPIX */
    {0.0, 0.0, 0.0, 0.0, 0.0, NULL, NULL, 0.0, 0, 0.0},   /* PRECIPPIX */
    {{0.0, 0.0}, {0.0, 0.0}, {0.0, 0.0}, 0.0, 0.0, 0.0}, /* PIXRAD */
    {0.0, 0.0},                                            /* RADCLASSPIX */
    {0.0, 0.0, 0, NULL, NULL, 0.0, 0, 0.0, 0.0, 0.0, 0.0, NULL, NULL, NULL,
     NULL, NULL, NULL, 0.0}, /* ROADSTRUCT*/
    {0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
     0.0}, /* SNOWPIX */
    {0,    0.0, NULL, NULL, NULL, 0.0, 0.0, 0.0, 0.0, 0.0,
     0.0, 0.0, 0.0,  0.0,  0.0,  0.0, 0.0, 0.0, 0.0, 0.0,
     0.0, 0.0, 0.0,  0.0,  0.0,  0.0, 0.0, 0.0},          /*SOILPIX */
    {0.0, 0.0, 0.0, 0.0, 0.0},                             /*SEDPIX */
    {0.0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}, /*FINEPIX */
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    01,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
```

```
    0.0,
    0.0,
    0.0};
CHANNEL ChannelData = {NULL, NULL, NULL, NULL, NULL, NULL, NULL,
                       NULL, NULL, NULL, NULL, NULL, NULL, NULL,
                       NULL, NULL, NULL, NULL, NULL, NULL, NULL,
                       NULL, NULL, NULL, NULL, NULL, NULL, NULL};
DUMPSTRUCT Dump = {0};
EVAPPIX **EvapMap = NULL;
INPUTFILES InFiles = {0};
LAYER Soil = {0};
LAYER Veg = {0};
LISTPTR Input = NULL; /* Linked list with input strings */
MAPSIZE Map = {0};          /* Size and location of model area */
MAPSIZE Radar = {0};  /* Size and location of area covered by precipitation radar */
MAPSIZE MM5Map = {0}; /* Size and location of area covered by MM5 input files */
METLOCATION *Stat = NULL;
OPTIONSTRUCT
    Options = {0}; /* Structure with information which program options to follow */
PIXMET LocalMet = {0}; /* Meteorological conditions for current pixel */
FINEPIX ***FineMap = NULL;
PRECIPPIX **PrecipMap = NULL;
RADARPIX **RadarMap = NULL;
RADCLASSPIX **RadMap = NULL;
PIXRAD **RadiationMap = NULL;
ROADSTRUCT **Network =
    NULL; /* 2D Array with channel information for each pixel */
SNOWPIX **SnowMap = NULL;
MET_MAP_PIX **MetMap = NULL;
SNOWTABLE *SnowAlbedo = NULL;
SOILPIX **SoilMap = NULL;
SEDPIX **SedMap = NULL;
SOILTABLE *SType = NULL;
SEDTABLE *SedType = NULL;
SOLARGEOMETRY SolarGeo = {0}; /* Geometry of Sun-Earth system (needed for INLINE
                         radiation calculations */
TIMESTRUCT Time = {0};
TOPOPIX **TopoMap = NULL;
UNITHYDR **UnitHydrograph = NULL;
UNITHYDRINFO HydrographInfo = {0}; /* Information about unit hydrograph */
VEGPIX **VegMap = NULL;
VEGTABLE *VType = NULL;
WATERBALANCE Mass = /* parameter for mass balance calculations */
    {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
     0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

/*****************************************************************************
  Initialization Procedures
*****************************************************************************/
if (argc < 2) {
  fprintf(stderr, "\nUsage: %s inputfile\n\n", argv[0]);
  fprintf(stderr, "DHSVM uses two output streams: \n");
  fprintf(stderr, "Standard Out, for the majority of output \n");
  fprintf(stderr, "Standard Error, for the final mass balance \n");
  fprintf(stderr, "\nTo pipe output correctly to files: \n");
```

```
  fprintf(stderr, "(cmd > f1) >& f2 \n");
  fprintf(stderr, "where f1 is stdout file and f2 is stderror file\n");
  exit(EXIT_FAILURE);
}

sprintf(commandline, "%s %s", argv[0], argv[1]);
printf("%s \n", commandline);
fprintf(stderr, "%s \n", commandline);
strcpy(InFiles.Const, argv[1]);

printf("\nRunning DHSVM %s\n", version);
printf("\nSTARTING INITIALIZATION PROCEDURES\n\n");

/* Start recording time */
start = clock();

ReadInitFile(InFiles.Const, &Input);
InitConstants(Input, &Options, &Map, &SolarGeo, &Time);

InitFileIO(Options.FileFormat);
InitTables(Time.NDaySteps, Input, &Options, &SType, &Soil, &VType, &Veg,
           &SnowAlbedo);

InitTerrainMaps(Input, &Options, &Map, &Soil, &TopoMap, &SoilMap, &VegMap);

CheckOut(Options.CanopyRadAtt, Veg, Soil, VType, SType, &Map, TopoMap, VegMap,
         SoilMap);

if (Options.HasNetwork)
  InitChannel(Input, &Map, Time.Dt, &ChannelData, SoilMap, &MaxStreamID,
              &MaxRoadID, &Options);
else if (Options.Extent != POINT)
  InitUnitHydrograph(Input, &Map, TopoMap, &UnitHydrograph, &Hydrograph,
                     &HydrographInfo);

InitNetwork(Map.NY, Map.NX, Map.DX, Map.DY, TopoMap, SoilMap, VegMap, VType,
            &Network, &ChannelData, Veg, &Options);

InitMetSources(Input, &Options, &Map, Soil.MaxLayers, &Time, &InFiles,
               &NStats, &Stat, &Radar, &MM5Map);

/* the following piece of code is for the UW PRISM project */
/* for real-time verification of SWE at Snotel sites */
/* Other users, set OPTION.SNOTEL to FALSE, or use TRUE with caution */

if (Options.Snotel == TRUE && Options.Outside == FALSE) {
  printf("Warning: All met stations locations are being set to the "
         "vegetation class GLACIER\n");
  printf("Warning: This requires that you have such a vegetation class in "
         "your vegetation table\n");
  printf("To disable this feature set Snotel OPTION to FALSE\n");
  for (int i = 0; i < NStats; i++) {
    printf("veg type for station %d is %d ", i,
           VegMap[Stat[i].Loc.N][Stat[i].Loc.E].Veg);
    for (j = 0; j < Veg.NTypes; j++) {
```

```
      if (VType[j].Index == GLACIER) {
        VegMap[Stat[i].Loc.N][Stat[i].Loc.E].Veg = j;
        break;
      }
    }
    if (j == Veg.NTypes) { /* glacier class not found */
      ReportError("MainDHSVM", 62);
    }
    printf("setting to glacier type (assumed bare class): %d\n", j);
  }
}

InitMetMaps(Time.NDaySteps, &Map, &Radar, &Options, InFiles.WindMapPath,
            InFiles.PrecipLapseFile, &PrecipLapseMap, &PrismMap, &ShadowMap,
            &SkyViewMap, &EvapMap, &PrecipMap, &RadarMap, &RadMap, SoilMap,
            &Soil, VegMap, &Veg, TopoMap, &MM5Input, &WindModel);

InitInterpolationWeights(&Map, &Options, TopoMap, &MetWeights, Stat, NStats);

InitDump(Input, &Options, &Map, Soil.MaxLayers, Veg.MaxLayers, Time.Dt,
         TopoMap, &Dump, &NGraphics, &which_graphics, id);

if (Options.HasNetwork == TRUE) {
  InitChannelDump(&Options, &ChannelData, Dump.Path);
  ReadChannelState(Dump.InitStatePath, &(Time.Start), ChannelData.streams);
}

InitSnowMap(&Map, &SnowMap);
InitAggregated(Veg.MaxLayers, Soil.MaxLayers, &Total);

InitModelState(&(Time.Start), &Map, &Options, PrecipMap, SnowMap, SoilMap,
               Soil, SType, VegMap, Veg, VType, Dump.InitStatePath,
               SnowAlbedo, TopoMap, Network, &HydrographInfo, Hydrograph);

InitNewMonth(&Time, &Options, &Map, TopoMap, PrismMap, ShadowMap, RadMap,
             &InFiles, Veg.NTypes, VType, NStats, Stat, Dump.InitStatePath);

InitNewDay(Time.Current.JDay, &SolarGeo);

if (NGraphics > 0) {
  printf("Initialzing X11 display and graphics \n");
  InitXGraphics(argc, argv, Map.NY, Map.NX, NGraphics, &MetMap);
}

shade_offset = FALSE;
if (Options.Shading == TRUE)
  shade_offset = TRUE;

/* Done with initialization, delete the list with input strings */
DeleteList(Input);

/*****************************************************************************
Sediment Initialization Procedures
*****************************************************************************/
if (Options.Sediment) {
```

```
  time(&tloc);
  srand(tloc);
  /* Randomize Random Generator */

  /* Commenting the line above and uncommenting the line below
     allows for the comparison of scenarios. */
  /*  srand48 (0);  */
  printf("\nSTARTING SEDIMENT INITIALIZATION PROCEDURES\n\n");

  ReadInitFile(Options.SedFile, &Input);

  InitParameters(Input, &Options, &Map, &Network, &ChannelData, TopoMap,
                 &Time, SedDiams);

  InitSedimentTables(Time.NDaySteps, Input, &SedType, &SType, &VType, &Soil,
                     &Veg);

  InitFineMaps(Input, &Options, &Map, &Soil, &TopoMap, &SoilMap, &FineMap);

  if (Options.HasNetwork) {
    printf("Initializing channel sediment\n\n");
    InitChannelSedimentDump(&ChannelData, Dump.Path, Options.ChannelRouting);
    InitChannelSediment(ChannelData.streams, &Total);
    InitChannelSediment(ChannelData.roads, &Total);
  }

  InitSedMap(&Map, &SedMap);

  /* Done with initialization, delete the list with input strings */
  DeleteList(Input);
}

/* setup for mass balance calculations */
Aggregate(&Map, &Options, TopoMap, &Soil, &Veg, VegMap, EvapMap, PrecipMap,
          RadMap, SnowMap, SoilMap, &Total, VType, Network, SedMap, FineMap,
          &ChannelData, &roadarea);

Mass.StartWaterStorage = Total.Soil.IExcess + Total.CanopyWater +
                         Total.SoilWater + Total.Snow.Swq +
                         Total.Soil.SatFlow;
Mass.OldWaterStorage = Mass.StartWaterStorage;

if (Options.Sediment) {
  Mass.StartChannelSedimentStorage = Total.ChannelSedimentStorage;
  Mass.LastChannelSedimentStorage = Mass.StartChannelSedimentStorage;
}

/* computes the number of grid cell contributing to one segment */
if (Options.StreamTemp)
  Init_segment_ncell(TopoMap, ChannelData.stream_map, Map.NY, Map.NX,
                     ChannelData.streams);

/*****************************************************************************
  Perform Calculations
*****************************************************************************/
```

```
while (Before(&(Time.Current), &(Time.End)) ||
       IsEqualTime(&(Time.Current), &(Time.End))) {
  PIXRAD localRadiation[T_COUNT] = {0};
  ResetAggregate(&Soil, &Veg, &Total, &Options);

  if (IsNewMonth(&(Time.Current), Time.Dt))
    InitNewMonth(&Time, &Options, &Map, TopoMap, PrismMap, ShadowMap, RadMap,
                 &InFiles, Veg.NTypes, VType, NStats, Stat,
                 Dump.InitStatePath);

  if (IsNewDay(Time.DayStep) && PRINT) {
    InitNewDay(Time.Current.JDay, &SolarGeo);
    PrintDate(&(Time.Current), stdout);
    printf("\n");
  }

  // determine surface erosion and routing scheme
  SedimentFlag(&Options, &Time);

  InitNewStep(&InFiles, &Map, &Time, Soil.MaxLayers, &Options, NStats, Stat,
              InFiles.RadarFile, &Radar, RadarMap, &SolarGeo, TopoMap, RadMap,
              SoilMap, MM5Input, WindModel, &MM5Map);

  // initialize channel/road networks for time step
  if (Options.HasNetwork) {
    channel_step_initialize_network(ChannelData.streams);
    channel_step_initialize_network(ChannelData.roads);
  }

  omp_set_num_threads(T_COUNT);
  #pragma omp parallel for collapse(2)
  for (int y = 0; y < Map.NY; y++) {
    for (int x = 0; x < Map.NX; x++) {
      int tid = omp_get_thread_num();
      PIXRAD *myRad = &(localRadiation[tid]);
      if (INBASIN(TopoMap[y][x].Mask)) {
        PIXMET LoopMet;
        if (Options.Shading)
          LoopMet = MakeLocalMetData(
              y, x, &Map, Time.DayStep, &Options, NStats, Stat,
              MetWeights[y][x], TopoMap[y][x].Dem, &(RadMap[y][x]),
              &(PrecipMap[y][x]), &Radar, RadarMap, PrismMap,
              &(SnowMap[y][x]), SnowAlbedo, MM5Input, WindModel,
              PrecipLapseMap, &MetMap, NGraphics, Time.Current.Month,
              SkyViewMap[y][x], ShadowMap[Time.DayStep][y][x],
              SolarGeo.SunMax, SolarGeo.SineSolarAltitude);
        else
          LoopMet = MakeLocalMetData(
              y, x, &Map, Time.DayStep, &Options, NStats, Stat,
              MetWeights[y][x], TopoMap[y][x].Dem, &(RadMap[y][x]),
              &(PrecipMap[y][x]), &Radar, RadarMap, PrismMap,
              &(SnowMap[y][x]), SnowAlbedo, MM5Input, WindModel,
              PrecipLapseMap, &MetMap, NGraphics, Time.Current.Month, 0.0,
              0.0, SolarGeo.SunMax, SolarGeo.SineSolarAltitude);
```

```
      for (int i = 0; i < Soil.MaxLayers; i++) {
        if (Options.HeatFlux == TRUE) {
          if (Options.MM5 == TRUE)
            SoilMap[y][x].Temp[i] =
                MM5Input[shade_offset + i + N_MM5_MAPS][y][x];
          else
            SoilMap[y][x].Temp[i] = Stat[0].Data.Tsoil[i];
        } else
          SoilMap[y][x].Temp[i] = LoopMet.Tair;
      }

      MassEnergyBalance(&Options, y, x, SolarGeo.SineSolarAltitude, Map.DX,
                        Map.DY, Time.Dt, Options.HeatFlux,
                        Options.CanopyRadAtt, Options.RoadRouting,
                        Options.Infiltration, Veg.MaxLayers, &LoopMet,
                        &(Network[y][x]), &(PrecipMap[y][x]),
                        &(VType[VegMap[y][x].Veg - 1]), &(VegMap[y][x]),
                        &(SType[SoilMap[y][x].Soil - 1]), &(SoilMap[y][x]),
                        &(SnowMap[y][x]), &(EvapMap[y][x]), myRad,
                        &ChannelData, SkyViewMap);

      PrecipMap[y][x].SumPrecip += PrecipMap[y][x].Precip;
    }
  }
}


for(int y = Map.NY - 1; y <= 0; y--) {
  int x;
  for(x = Map.NX - 1; x <= 0; x--) {
    if (INBASIN(TopoMap[y][x].Mask)) {
      if (Options.Shading)
        LocalMet = MakeLocalMetData(
            y, x, &Map, Time.DayStep, &Options, NStats, Stat,
            MetWeights[y][x], TopoMap[y][x].Dem, &(RadMap[y][x]),
            &(PrecipMap[y][x]), &Radar, RadarMap, PrismMap,
            &(SnowMap[y][x]), SnowAlbedo, MM5Input, WindModel,
            PrecipLapseMap, &MetMap, NGraphics, Time.Current.Month,
            SkyViewMap[y][x], ShadowMap[Time.DayStep][y][x],
            SolarGeo.SunMax, SolarGeo.SineSolarAltitude);
      else
        LocalMet = MakeLocalMetData(
            y, x, &Map, Time.DayStep, &Options, NStats, Stat,
            MetWeights[y][x], TopoMap[y][x].Dem, &(RadMap[y][x]),
            &(PrecipMap[y][x]), &Radar, RadarMap, PrismMap,
            &(SnowMap[y][x]), SnowAlbedo, MM5Input, WindModel,
            PrecipLapseMap, &MetMap, NGraphics, Time.Current.Month, 0.0,
            0.0, SolarGeo.SunMax, SolarGeo.SineSolarAltitude);
      break;
    }
  }
  if (INBASIN(TopoMap[y][x].Mask))
    break;
}
```

```
    for(int i = 0; i < T_COUNT; i ++) {
      PIXRAD *Rad = &(localRadiation[i]);
      for (int j = 0; j < Veg.MaxLayers + 1; j++) {
        Total.Rad.NetShort[j] += Rad->NetShort[j];
        Total.Rad.LongIn[j] += Rad->LongIn[j];
        Total.Rad.LongOut[j] += Rad->LongOut[j];
      }
      Total.Rad.PixelNetShort += Rad->PixelNetShort;
      Total.Rad.PixelLongIn += Rad->PixelLongIn;
      Total.Rad.PixelLongOut += Rad->PixelLongOut;
    }

    // Average all RBM inputs over each segment
    if (Options.StreamTemp) {
      channel_grid_avg(ChannelData.streams);
      if (Options.CanopyShading)
        CalcCanopyShading(ChannelData.streams, &SolarGeo);
    }

#ifndef SNOW_ONLY

    // set sediment inflows to zero - they are incremented elsewhere
    if ((Options.HasNetwork) && (Options.Sediment)) {
      InitChannelSedInflow(ChannelData.streams);
      InitChannelSedInflow(ChannelData.roads);
    }

    RouteSubSurface(Time.Dt, &Map, TopoMap, VType, VegMap, Network, SType,
                    SoilMap, &ChannelData, &Time, &Options, Dump.Path, SedMap,
                    FineMap, SedType, MaxStreamID, SnowMap);

    if (Options.HasNetwork)
      RouteChannel(&ChannelData, &Time, &Map, TopoMap, SoilMap, &Total,
                   &Options, Network, SType, PrecipMap, SedMap, LocalMet.Tair,
                   LocalMet.Rh, SedDiams);

    // Sediment Routing in Channel and output to sediment files
    if ((Options.HasNetwork) && (Options.Sediment)) {
      //SPrintDate(&(Time.Current), buffer);
      flag = IsEqualTime(&(Time.Current), &(Time.Start));

      if (Options.ChannelRouting) {
        if (ChannelData.roads != NULL) {
          RouteChannelSediment(ChannelData.roads, Time, &Dump, &Total,
                               SedDiams);
          channel_save_sed_outflow_text(buffer, ChannelData.roads,
                                        ChannelData.sedroadout,
                                        ChannelData.sedroadflowout, flag);
          RouteCulvertSediment(&ChannelData, &Map, TopoMap, SedMap, &Total,
                               SedDiams);
        }
        RouteChannelSediment(ChannelData.streams, Time, &Dump, &Total,
                             SedDiams);
        channel_save_sed_outflow_text(buffer, ChannelData.streams,
                                      ChannelData.sedstreamout,
```

```
                                        ChannelData.sedstreamflowout, flag);
    } else {
      if (ChannelData.roads != NULL) {
        channel_save_sed_inflow_text(buffer, ChannelData.roads,
                                     ChannelData.sedroadinflow, SedDiams,
                                     flag);
      }
      channel_save_sed_inflow_text(buffer, ChannelData.streams,
                                   ChannelData.sedstreaminflow, SedDiams,
                                   flag);
    }
    SaveChannelSedInflow(ChannelData.roads, &Total);
    SaveChannelSedInflow(ChannelData.streams, &Total);
  }

  if (Options.Extent == BASIN)
    RouteSurface(&Map, &Time, TopoMap, SoilMap, &Options, UnitHydrograph,
                 &HydrographInfo, Hydrograph, &Dump, VegMap, VType, SType,
                 &ChannelData, SedMap, PrecipMap, SedType, LocalMet.Tair,
                 LocalMet.Rh, SedDiams);

#endif

  if (NGraphics > 0)
    draw(&(Time.Current), IsEqualTime(&(Time.Current), &(Time.Start)),
         Time.DayStep, &Map, NGraphics, which_graphics, VType, SType, SnowMap,
         SoilMap, SedMap, FineMap, VegMap, TopoMap, PrecipMap, PrismMap,
         SkyViewMap, ShadowMap, EvapMap, RadMap, MetMap, Network, &Options);

  Aggregate(&Map, &Options, TopoMap, &Soil, &Veg, VegMap, EvapMap, PrecipMap,
            RadMap, SnowMap, SoilMap, &Total, VType, Network, SedMap, FineMap,
            &ChannelData, &roadarea);

  MassBalance(&(Time.Current), &(Dump.Balance), &(Dump.SedBalance), &Total,
              &Mass, &Options);

  ExecDump(&Map, &(Time.Current), &(Time.Start), &Options, &Dump, TopoMap,
           EvapMap, RadiationMap, PrecipMap, RadMap, SnowMap, MetMap, VegMap,
           &Veg, SoilMap, SedMap, Network, &ChannelData, FineMap, &Soil,
           &Total, &HydrographInfo, Hydrograph);

  IncreaseTime(&Time);
  t += 1;
}

ExecDump(&Map, &(Time.Current), &(Time.Start), &Options, &Dump, TopoMap,
         EvapMap, RadiationMap, PrecipMap, RadMap, SnowMap, MetMap, VegMap,
         &Veg, SoilMap, SedMap, Network, &ChannelData, FineMap, &Soil, &Total,
         &HydrographInfo, Hydrograph);
writeRandomVals(Dump.Path);
FinalMassBalance(&(Dump.FinalBalance), &Total, &Mass, &Options, roadarea);

/*printf("\nSTARTING CLEANUP\n\n");*/
cleanup(&Dump, &ChannelData, &Options);
printf("\nEND_OF_MODEL_RUN\n\n");
```

```
  /* record the run time at the end of each time loop */
  finish1 = clock();
  runtime = (finish1 - start) / CLOCKS_PER_SEC;
  printf("*************************************************************************"
         "**************");
  printf("\nRuntime Summary:\n");
  printf("%6.2f hours elapsed for the simulation period of %d hours (%.1f "
         "days) \n",
         runtime / 3600, t * Time.Dt / 3600, (float)t * Time.Dt / 3600 / 24);

  clock_gettime(CLOCK_MONOTONIC, &tfinish);
  elapsed = (tfinish.tv_sec - tstart.tv_sec);
  elapsed += (tfinish.tv_nsec - tstart.tv_nsec) / 1000000000.0;
  fprintf(stderr, "\n\n\n");
  fprintf(stderr, "\n\nRan for %lf seconds\n", elapsed);
  return EXIT_SUCCESS;
}
/*****************************************************************************
  Cleanup
*****************************************************************************/
void cleanup(DUMPSTRUCT *Dump, CHANNEL *ChannelData, OPTIONSTRUCT *Options) {
  if (Dump->Aggregate.FilePtr != NULL)
    fclose(Dump->Aggregate.FilePtr);
  if (Dump->Balance.FilePtr != NULL)
    fclose(Dump->Balance.FilePtr);
  if (Dump->FinalBalance.FilePtr != NULL)
    fclose(Dump->FinalBalance.FilePtr);
  if (ChannelData->streamflowout != NULL)
    fclose(ChannelData->streamflowout);
  if (ChannelData->streamout != NULL)
    fclose(ChannelData->streamout);
  if (ChannelData->roadflowout != NULL)
    fclose(ChannelData->roadflowout);
  if (ChannelData->roadout != NULL)
    fclose(ChannelData->roadout);

  if (Options->StreamTemp) {
    if (ChannelData->streaminflow != NULL)
      fclose(ChannelData->streaminflow);
    if (ChannelData->streamoutflow != NULL)
      fclose(ChannelData->streamoutflow);
    if (ChannelData->streamISW != NULL)
      fclose(ChannelData->streamISW);
    if (ChannelData->streamNSW != NULL)
      fclose(ChannelData->streamNSW);
    if (ChannelData->streamILW != NULL)
      fclose(ChannelData->streamILW);
    if (ChannelData->streamNLW != NULL)
      fclose(ChannelData->streamNLW);
    if (ChannelData->streamVP != NULL)
      fclose(ChannelData->streamVP);
    if (ChannelData->streamWND != NULL)
      fclose(ChannelData->streamWND);
    if (ChannelData->streamATP != NULL)
```

```
        fclose(ChannelData->streamATP);
      if (ChannelData->streamBeam != NULL)
        fclose(ChannelData->streamBeam);
      if (ChannelData->streamDiffuse != NULL)
        fclose(ChannelData->streamDiffuse);
      if (ChannelData->streamSkyView != NULL)
        fclose(ChannelData->streamSkyView);
    }
}
```

Appendix C

INPUT FOR DHSVM

Code Listing C.1: Input file for DHSVM

```
################################################################################
# DHSVM INPUT FILE FORMAT
################################################################################
# The file is organized in sections [...], which contain key = entry pairs.
# The file is free format, in that correct reading of the file is not dependent
# on spaces and/or the order of the key-entry pairs within a section.
# The keys are not case-sensitive, but the entries are, because filenames on a
# UNIX platform are case-sensitive.
# Comments are preceded by a '#', and run from the occurrence of '#' till the
# end of the line. You can comment out an entire line (like in this
# header), or you can place a comment after an entry.
# It is important to place the key-entry pair in the correct section, since it
# will not be found if it is in another section.
# The easiest way to make the input file is to fill out this default template.
# Since DHSVM will only use the keys that it requires you do not have to worry
# about empty entries for keys that are not needed. For example, if you are
# running the model in point mode, you do not have to fill out the routing
# section.  If you have already filled it out you can leave it, since DHSVM will
# not use the information.  This allows easy switching between point and basin
# mode.
# For more information about the specific entries see the DHSVM web page

################################################################################
# OPTIONS SECTION
################################################################################
[OPTIONS]                                  # Model Options
Format                 = BIN               # BIN or NETCDF
Extent                 = BASIN             # POINT or BASIN
Gradient               = WATERTABLE              # TOPOGRAPHY or WATERTABLE
Flow Routing           = NETWORK           # UNIT_HYDROGRAPH or NETWORK
Sensible Heat Flux     = FALSE             # TRUE or FALSE
Sediment                 = FALSE           # TRUE or FALSE
Sediment Input File    =                   # path for sediment configuration file
Overland Routing       = CONVENTIONAL      # CONVENTIONAL or KINEMATIC
Infiltration             = STATIC          # Static or Dynamic
Interpolation          = VARCRESS          # NEAREST or INVDIST or VARCRESS
MM5                    = FALSE             # TRUE or FALSE
QPF                    = FALSE             # TRUE or FALSE
PRISM                  = FALSE             # TRUE or FALSE
PRISM data path        =                   # path for PRISM files
PRISM data extension = bin                 # file extension for PRISM files
Canopy radiation attenuation mode = FIXED # FIXED or VARIABLE
Shading                = FALSE                   # TRUE or FALSE
Shading data path      = ../input/Shadow
Shading data extension = nc                # file extension for shading files
Skyview data path      = ../input/SkyView.bin
```

```
Snotel                = FALSE              # TRUE or FALSE
Outside               = TRUE                   # TRUE or FALSE
Rhoverride            = FALSE              # TRUE or FALSE
Precipitation Source = STATION             # STATION or RADAR
Wind Source           = STATION            # STATION or MODEL
Temperature lapse rate    = CONSTANT       # CONSTANT or VARIABLE
Precipitation lapse rate = CONSTANT        # CONSTANT, MAP, or VARIABLE
Cressman radius       = 10                 # in model pixels
Cressman stations     =  4                 # number of stations
Stream Temperature    = TRUE               # TRUE or FALSE
Canopy Shading        = FALSE

##############################################################################
# MODEL AREA SECTION
##############################################################################
[AREA]                                     # Model area
Coordinate System     = UTM                # UTM or USER_DEFINED
Extreme North         = 5279315.465411     # Coordinate for northern edge of grid
Extreme West          = 560662.021456      # Coordinate for western edge of grid
Center Latitude       = 47.60893338077     # Central parallel of basin
Center Longitude      = -122.1507756929    # Central meridian of basin
Time Zone Meridian    = -120.0             # Time zone meridian for area
Number of Rows        = 423                # Number of rows
Number of Columns     = 211                # Number of columns
Grid spacing          = 30                 # Grid resolution in m
Point North           =                    # North coordinate for point model if Extent = POINT
Point East            =                    # East coordinate for point modelif Extent = POINT

##############################################################################
# TIME SECTION
##############################################################################
[TIME]                                     # Model period
Time Step             = 3                  # Model time step (hours)
Model Start           = 01/01/2002-03      # Model start time (MM/DD/YYYY-HH)
Model End             = 04/02/2004-06      # Model end time (MM/DD/YYYY-HH)

##############################################################################
# CONSTANTS SECTION
##############################################################################
[CONSTANTS]                                # Model constants
Ground Roughness      = 0.02               # Roughness of soil surface (m)
Snow Roughness        = 0.01               # Roughness of snow surface (m)
Rain Threshold        = -1.0               # Minimum temperature at which rain occurs (C)
Snow Threshold        = 0.5                # Maximum temperature at which snow occurs (C)
Snow Water Capacity   = 0.03               # Snow liquid water holding capacity(fraction)
Reference Height      = 45.0               # Reference height (m)
Rain LAI Multiplier   = 0.0001             # LAI Multiplier for rain interception
Snow LAI Multiplier   = 0.0005             # LAI Mulitplier for snow interception
Min Intercepted Snow = 0.005               # Intercepted snow that can only be melted (m)
Outside Basin Value   = 0                  # Value in mask that indicates outside the basin
Temperature Lapse Rate    = -0.006         # Temperature lapse rate (C/m)
Precipitation Lapse Rate =  0.0001         # Precipitation lapse rate (m/m)
Precipitation Multiplier = 0.

####################Only if CanopyShading is TRUE#################################
```

```
Tree Height =
Buffer Width =
Overhang Coefficient =
Monthly Extinction Coefficient = 0. 0. 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0. 0.
                                        # extinction coefficient (0 ~ 1)
Canopy Bank Distance =                  # distance from bank to canopy (m)

##############################################################################
# TERRAIN INFORMATION SECTION
##############################################################################
[TERRAIN]                               # Terrain information

DEM File            = ../input/dem.bin
Basin Mask File     = ../input/mask.bin
##############################################################################
# ROUTING SECTION
##############################################################################
[ROUTING]                               # Routing information. This section is
                                        # only relevant if the Extent = BASIN
################## STREAM NETWORK ############################################
# The following three fields are only used if Flow Routing = NETWORK

Stream Map File       = ../input/stream.map
Stream Network File   = ../input/stream.network
Stream Class File     = ../input/adjust.classfile

################################### ROAD NETWORK #############################
# The following three fields are only used if Flow Routing = NETWORK and there
# is a road network

#Road Map File         =                 # path for road map file
#Road Network File     =                 # path for road network file
#Road Class File       =                 # path for road network file

######################### UNIT HYDROGRAPH ####################################
# The following two fields are only used if Flow Routing = UNIT_HYDROGRAPH

Travel Time File      =                 # path for travel time file
Unit Hydrograph File =                  # path for unit hydrograph file

##############################################################################
# METEOROLOGY SECTION
##############################################################################
[METEOROLOGY]                           # Meteorological station

Number of Stations = 2                  # Number of meteorological stations

Station Name 1      = VIC_pseudo_station01
North Coordinate 1 = 5271296
East Coordinate 1   = 563526.75
Elevation 1         = 82.07555
Station File 1      = ../met/met.met.met07_windprofile

Station Name 2     = VIC_pseudo_station02
North Coordinate 2 = 5278295
```

```
East Coordinate 2  = 568144.3125
Elevation 2        = 65.80444
Station File 2     = ../met/met.met.met09_windprofile

##################################### MM5 ########################################
# The following block only needs to be filled out if MM5 = TRUE.  In that case
# This is the ONLY block that needs to be filled out

MM5 Start              =                  # Start of MM5 file (MM/DD/YYYY-HH),
MM5 Rows               =
MM5 Cols               =
MM5 Extreme North      =
MM5 Extreme West       =
MM5 DY                 =

# MM5 met files
MM5 Temperature File   =
MM5 Humidity File      =
MM5 Wind Speed File    =
MM5 Shortwave File     =
MM5 Longwave File      =
MM5 Pressure File      =
MM5 Precipitation File =
MM5 Terrain File       =
MM5 Temp Lapse File    =

# For each soil layer make a key-entry pair as below (n = 1, ..,
# Number of Soil Layers)

MM5 Soil Temperature File 0 =
MM5 Soil Temperature File 1 =
MM5 Soil Temperature File 2 =

################################## RADAR #########################################
# The following block only needs to be filled out if Precipitation Source =
# RADAR.
Radar Start            =
Radar File             =
Radar Extreme North    =
Radar Extreme West     =
Radar Number of Rows   =
Radar Number of Columns =
Radar Grid Spacing     =

################################## Wind ##########################################
# The following block only needs to be filled out if Wind Source = MODEL
Number of Wind Maps    =
Wind File Basename     =
Wind Map Met Stations  =


###################### Precipitation lapse rate ##################################
# The following block only needs to be filled out if Precipitation lapse rate
# = MAP
Precipitation lapse rate =
```

```
#############################################################################
# SOILS INFORMATION SECTION
#############################################################################
[SOILS]                                  # Soil information
Soil Map File    = ../input/soil.bin
Soil Depth File = ../input/soild.bin

Number of Soil Types = 4

################################## SOIL 1 ###################################
Soil Description        1 = SAND
Lateral Conductivity    1 = 0.1
Exponential Decrease    1 = 3.0
Depth Threshold         1 = 0.5
Maximum Infiltration    1 = 1.8e-5

Capillary Drive         1 =
Surface Albedo          1 = 0.1
Number of Soil Layers   1 = 3
Porosity                1 =  .43 .43 .43
Pore Size Distribution  1 =  .24 .24 .24
Bubbling Pressure       1 =  .07 .07 .07
Field Capacity          1 =  .08 .08 .08
Wilting Point           1 =  .03 .03 .03
Bulk Density            1 = 1492. 1492. 1492.
Vertical Conductivity   1 = 5.0E-1 5.0E-1 5.0E-1
Thermal Conductivity    1 = 7.114  6.923 6.923
Thermal Capacity        1 = 1.4e6  1.4e6 1.4e6
Mannings n              1 =

################################## SOIL 2 ###################################
Soil Description        2 = LOAMY SAND
Lateral Conductivity    2 = 0.03
Exponential Decrease    2 = 3.5
Maximum Infiltration    2 = 3.0e-6
Depth Threshold         2 = 0.5
Capillary Drive         2 =
Surface Albedo          2 = 0.1
Number of Soil Layers   2 = 3
Porosity                2 =  .46 .46 .46
Pore Size Distribution  2 =  .26 .26 .26
Bubbling Pressure       2 =  .21 .21 .21
Field Capacity          2 =  .38 .38 .38
Wilting Point           2 =  .047 .047 .047
Bulk Density            2 = 1419. 1419. 1419.
Vertical Conductivity   2 = 2.2E-6 2.2E-5 2.2E-6
Thermal Conductivity    2 = 7.114  6.923 7.0
Thermal Capacity        2 = 1.4e6  1.4e6 1.4e6
Mannings n              2 =

################################## SOIL 3 ###################################
Soil Description        3 = SANDY LOAM
Lateral Conductivity    3 = 7.0e-4
Exponential Decrease    3 = 3.0
```

```
Maximum Infiltration    3 = 8.0e-6
Depth Threshold         3 = 0.5
Capillary Drive         3 =
Surface Albedo          3 = 0.1
Number of Soil Layers   3 = 3
Porosity                3 = .46 .46 .46
Pore Size Distribution  3 = .21 .21 .21
Bubbling Pressure       3 = .21 .21 .21
Field Capacity          3 = .15 .15 .15
Wilting Point           3 = 1E-3 1E-3 1E-3
Bulk Density            3 = 1419. 1419. 1419.
Vertical Conductivity   3 = 0.07 0.07 0.07
Thermal Conductivity    3 = 7.114  6.923 7.0
Thermal Capacity        3 = 1.4e6  1.4e6 1.4e6
Mannings n              3 =

################################ SOIL 4 ##########################################
Soil Description        4 = SANDY LOAM
Lateral Conductivity    4 = 0.004
Exponential Decrease    4 = 1.0
Maximum Infiltration    4 = 1e-3
Depth Threshold         4 = 0.5
Capillary Drive         4 =
Surface Albedo          4 = 0.1
Number of Soil Layers   4 = 3
Porosity                4 =  .42 .42 .42
Pore Size Distribution  4 =  .21 .21 .21
Bubbling Pressure       4 =  .15 .15 .15
Field Capacity          4 =  .25 .25 .25
Wilting Point           4 =  .12 .12 .12
Bulk Density            4 = 1569. 1569. 1569.
Vertical Conductivity   4 = 3.0E-3 3.0E-3 3.0E-3
Thermal Conductivity    4 = 7.114  6.923 7.0
Thermal Capacity        4 = 1.4e6  1.4e6 1.4e6
Mannings n              4 =

##############################################################################
# VEGETATION INFORMATION SECTION
##############################################################################
[VEGETATION]

Vegetation Map File = ../input/veg.bin

Number of Vegetation Types = 14          # Number of different vegetation types
######################## VEGETATION 1 ########################################
Vegetation Description    1 = Dense Urban (>75%)
Impervious Fraction       1 = 0.85
Detention Fraction        1 = 0
Detention Decay           1 = 0
Overstory Present         1 = FALSE
Understory Present        1 = TRUE
Fractional Coverage       1 =
Hemi Fract Coverage       1 =
Trunk Space               1 =
Aerodynamic Attenuation   1 =
```

```
Radiation Attenuation     1 =
Clumping Factor           1 =             /* Required if CanopyRadiationAttenuation==VARIABLE */
Leaf Angle A              1 =             /* Required if ..................................... */
Leaf Angle B              1 =             /* Required if ..................................... */
Scattering Parameter      1 =             /* Required if ..................................... */
Max Snow Int Capacity     1 =
Snow Interception Eff     1 =
Mass Release Drip Ratio   1 =
IMPERVIOUS SURFACE ROUTING FILE = ../input/surface.routing.txt
Height                    1 = 0.2
Overstory Monthly LAI     1 =
Understory Monthly LAI    1 = 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0
Maximum Resistance        1 = 3000.
Minimum Resistance        1 = 340.
Moisture Threshold        1 = 0.33
Vapor Pressure Deficit    1 = 4000
Rpc                       1 = .108
Overstory Monthly Alb     1 =
Understory Monthly Alb    1 = 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4
Number of Root Zones      1 = 3
Root Zone Depths          1 = 0.10 0.25 0.40
Overstory Root Fraction   1 = 0.20 0.40 0.40
Understory Root Fraction  1 = 0.40 0.60 0.00

############################ VEGETATION 2 ##########################################
Vegetation Description    2 = Light/medium Urban (<75%)
Impervious Fraction       2 = 0.25
Detention Fraction        2 = 0
Detention Decay           2 = 0
IMPERVIOUS SURFACE ROUTING FILE = ../input/surface.routing.txt
Overstory Present         2 = FALSE
Understory Present        2 = TRUE
Fractional Coverage       2 =
Hemi Fract Coverage       2 =
Clumping Factor           2 =
Leaf Angle A              2 =
Leaf Angle B              2 =
Scattering Parameter      2 =
Trunk Space               2 =
Aerodynamic Attenuation   2 =
Radiation Attenuation     2 =
Max Snow Int Capacity     2 =
Snow Interception Eff     2 =
Mass Release Drip Ratio   2 =
Height                    2 = 0.2
Overstory Monthly LAI     2 =
Understory Monthly LAI    2 = 3.0 3.0 2.0 3.0 3.0 2.0 3.0 3.0 2.0 3.0 3.0 2.0
Maximum Resistance        2 = 3000.
Minimum Resistance        2 = 300
Moisture Threshold        2 = 0.33
Vapor Pressure Deficit    2 = 4000
Rpc                       2 = 0.108
Overstory Monthly Alb     2 =
Understory Monthly Alb    2 = 0.20 0.20 0.20 0.20 0.20 0.20 0.20 0.20 0.20 0.20 0.20 0.20
Number of Root Zones      2 = 3
```

```
Root Zone Depths          2 = 0.10  0.25  0.40
Overstory Root Fraction   2 = 0.20  0.40  0.40
Understory Root Fraction  2 = 0.40  0.60  0.00

############################ VEGETATION 3 ##########################################
Vegetation Description    3 = Bareground
Impervious Fraction       3 = 0.0
Detention Fraction        3 = 0.0
Detention Decay           3 = 0.0
Overstory Present         3 = FALSE
Understory Present        3 = FALSE
Fractional Coverage       3 =
Hemi Fract Coverage       3 =
Clumping Factor           3 =
Leaf Angle A              3 =
Leaf Angle B              3 =
Scattering Parameter      3 =
Trunk Space               3 =
Aerodynamic Attenuation   3 =
Radiation Attenuation     3 =
Max Snow Int Capacity     3 =
Snow Interception Eff     3 =
Mass Release Drip Ratio   3 =
Height                    3 =
Overstory Monthly LAI     3 =
Understory Monthly LAI    3 =
Maximum Resistance        3 =
Minimum Resistance        3 =
Moisture Threshold        3 =
Vapor Pressure Deficit    3 =
Rpc                       3 =
Overstory Monthly Alb     3 =
Understory Monthly Alb    3 =
Number of Root Zones      3 = 3
Root Zone Depths          3 = 0.10  0.25  0.40
Overstory Root Fraction   3 = 0.20  0.40  0.40
Understory Root Fraction  3 = 0.40  0.60  0.00

############################ VEGETATION 4 ##########################################
Vegetation Description    4 = Dry Ground
Impervious Fraction       4 = 0.0
Detention Fraction        4 = 0.0
Detention Decay           4 = 0.0
Overstory Present         4 = FALSE
Understory Present        4 = TRUE
Fractional Coverage       4 =
Hemi Fract Coverage       4 =
Clumping Factor           4 =
Leaf Angle A              4 =
Leaf Angle B              4 =
Scattering Parameter      4 =
Trunk Space               4 = 0.38
Aerodynamic Attenuation   4 =
Radiation Attenuation     4 =
Max Snow Int Capacity     4 =
```

```
Snow Interception Eff     4 =
Mass Release Drip Ratio   4 =
Height                    4 = 0.5
Overstory Monthly LAI     4 =
Understory Monthly LAI    4 = 5.0 2.0 2.0 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5
Maximum Resistance        4 = 600.
Minimum Resistance        4 = 320
Moisture Threshold        4 = 0.33
Vapor Pressure Deficit    4 = 4000
Rpc                       4 = 0.108
Overstory Monthly Alb     4 =
Understory Monthly Alb    4 = 0.19 0.19 0.9 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19
Number of Root Zones      4 = 3
Root Zone Depths          4 = 0.10 0.25 0.40
Overstory Root Fraction   4 = 0.20 0.40 0.40
Understory Root Fraction  4 = 0.40 0.60 0.00

############################ VEGETATION 5 ##########################################
Vegetation Description    5 = Native Grass
Impervious Fraction       5 = 0.0
Detention Fraction        5 = 0.0
Detention Decay           5 = 0.0
Overstory Present         5 = TRUE
Understory Present        5 = TRUE
Fractional Coverage       5 = 0.5
Hemi Fract Coverage       5 = 0.5
Clumping Factor           5 =
Leaf Angle A              5 =
Leaf Angle B              5 =
Scattering Parameter      5 =
Trunk Space               5 = 0.4
Aerodynamic Attenuation   5 = 0.3
Radiation Attenuation     5 = 0.1
Max Snow Int Capacity     5 = 0.003
Snow Interception Eff     5 = 0.6
Mass Release Drip Ratio   5 = 0.4
Height                    5 = 20.0 0.5
Overstory Monthly LAI     5 = 2.0 2.0 2.0 2.0 2.0 6.0 6.0 6.0 6.0 2.0 2.0 2.0
Understory Monthly LAI    5 = 1.0 1.0 1.0 1.0 1.0 3.3 3.3 3.3 3.3 1.0 1.0 1.0
Maximum Resistance        5 = 1000 600.
Minimum Resistance        5 = 280 200.
Moisture Threshold        5 = 0.33 0.33
Vapor Pressure Deficit    5 = 4000  4000
Rpc                       5 = 0.108 0.108
Overstory Monthly Alb     5 = 0.15 0.15 0.15 0.15 0.15 0.14 0.14 0.14 0.14 0.15 0.15 0.15
Understory Monthly Alb    5 = 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18 0.18
Number of Root Zones      5 = 3
Root Zone Depths          5 = 0.10 0.25 0.40
Overstory Root Fraction   5 = 0.20 0.40 0.40
Understory Root Fraction  5 = 0.40 0.60 0.00

############################ VEGETATION 6 ##########################################
Vegetation Description    6 = Grass/crop/shrub
Impervious Fraction       6 = 0.0
Detention Fraction        6 = 0.0
```

```
Detention Decay           6 = 0.0
Overstory Present         6 = FALSE
Understory Present        6 = TRUE
Fractional Coverage       6 =
Hemi Fract Coverage       6 =
Clumping Factor           6 =
Leaf Angle A              6 =
Leaf Angle B              6 =
Scattering Parameter      6 =
Trunk Space               6 =
Aerodynamic Attenuation   6 =
Radiation Attenuation     6 =
Max Snow Int Capacity     6 =
Snow Interception Eff     6 =
Mass Release Drip Ratio   6 =
Height                    6 = 1.0
Overstory Monthly LAI     6 =
Understory Monthly LAI    6 = 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0
Maximum Resistance        6 = 600.
Minimum Resistance        6 = 280
Moisture Threshold        6 = 0.33
Vapor Pressure Deficit    6 = 4000
Rpc                       6 = .108
Overstory Monthly Alb     6 =
Understory Monthly Alb    6 = 0.24 0.24 0.23 0.22 0.21 0.20 0.20 0.20 0.22 0.23 0.24 0.24
Number of Root Zones      6 = 3
Root Zone Depths          6 = 0.10 0.25 0.40
Overstory Root Fraction   6 = 0.20 0.40 0.40
Understory Root Fraction  6 = 0.40 0.60 0.00

############################ VEGETATION 7 ##########################################
Vegetation Description    7 = Mixed/deciduous Forest
Impervious Fraction       7 = 0.0
Detention Fraction        7 = 0
Detention Decay           7 = 0
Overstory Present         7 = TRUE
Understory Present        7 = TRUE
Fractional Coverage       7 = 0.85
Hemi Fract Coverage       7 = 0.85
Clumping Factor           7 =
Leaf Angle A              7 =
Leaf Angle B              7 =
Scattering Parameter      7 =
Trunk Space               7 = 0.45
Aerodynamic Attenuation   7 = 2.0
Radiation Attenuation     7 = 0.1875
Max Snow Int Capacity     7 = 0.01225
Snow Interception Eff     7 = 0.6
Mass Release Drip Ratio   7 = 0.4
Height                    7 = 25.0 0.5
Overstory Monthly LAI     7 = 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0
Understory Monthly LAI    7 = 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0
Maximum Resistance        7 = 1500. 1000.
Minimum Resistance        7 = 666.6 823.4
Moisture Threshold        7 = 0.33 0.13
```

```
Vapor Pressure Deficit    7 = 4000 4000
Rpc                       7 = .108 .108
Overstory Monthly Alb     7 = 0.27 0.27 0.25 0.25 0.22 0.21 0.21 0.21 0.21 0.22 0.24 0.26
Understory Monthly Alb    7 = 0.175 0.175 0.175 0.175 0.175 0.175 0.175 0.175 0.175 0.175 0.175
    0.175
Number of Root Zones      7 = 3

Root Zone Depths          7 = 0.10 0.25 0.40
Overstory Root Fraction   7 = 0.20 0.40 0.40
Understory Root Fraction  7 = 0.40 0.60 0.00

############################ VEGETATION 8 ########################################
Vegetation Description    8 = Conifer Forest
Impervious Fraction       8 = 0.0
Detention Fraction        8 = 0
Detention Decay           8 = 0
Overstory Present         8 = TRUE
Understory Present        8 = TRUE
Fractional Coverage       8 = 0.9
Hemi Fract Coverage       8 = 0.9
Clumping Factor           8 =
Leaf Angle A              8 =
Leaf Angle B              8 =
Scattering Parameter      8 =
Trunk Space               8 = 0.5
Aerodynamic Attenuation   8 = 2.5
Radiation Attenuation     8 = 0.16
Max Snow Int Capacity     8 = 0.03
Snow Interception Eff     8 = 0.6
Mass Release Drip Ratio   8 = 0.4
Height                    8 = 43.3 0.5
Overstory Monthly LAI     8 = 12 12 12 12 12 12 12 12 12 12 12 12
Understory Monthly LAI    8 = 6 6 6 6 6 6 6 7 7 6 6 6
Maximum Resistance        8 = 2000 2000
Minimum Resistance        8 = 1333.2 855.54
Moisture Threshold        8 = 0.33 0.13
Vapor Pressure Deficit    8 = 4000 4000
Rpc                       8 = .108 .108
Overstory Monthly Alb     8 = 0.14 0.14 0.14 0.13 0.13 0.12 0.11 0.11 0.12 0.13 0.14 0.14
Understory Monthly Alb    8 = 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19
Number of Root Zones      8 = 3
Root Zone Depths          8 = 0.10 0.25 0.40
Overstory Root Fraction   8 = 0.20 0.40 0.40
Understory Root Fraction  8 = 0.40 0.60 0.00

############################ VEGETATION 9 ########################################
Vegetation Description    9 = Regrowth Vegetation
Impervious Fraction       9 = 0.0
Detention Fraction        9 = 0
Detention Decay           9 = 0
Overstory Present         9 = FALSE
Understory Present        9 = TRUE
Fractional Coverage       9 =
Hemi Fract Coverage       9 =
Clumping Factor           9 =
```

```
Leaf Angle A              9 =
Leaf Angle B              9 =
Scattering Parameter      9 =
Trunk Space               9 =
Aerodynamic Attenuation   9 =
Radiation Attenuation     9 =
Max Snow Int Capacity     9 =
Snow Interception Eff     9 =
Mass Release Drip Ratio   9 =
Height                    9 =  1.0
Overstory Monthly LAI     9 =
Understory Monthly LAI    9 = 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
Maximum Resistance        9 =  600
Minimum Resistance        9 =  220
Moisture Threshold        9 =  0.33
Vapor Pressure Deficit    9 =  4000
Rpc                       9 =  .108
Overstory Monthly Alb     9 =
Understory Monthly Alb    9 = 0.2 0.2 0.2 0.19 0.18 0.17 0.16 0.16 0.17 0.18 0.19 0.2
Number of Root Zones      9 = 3
Root Zone Depths          9 =  0.10 0.25 0.40
Overstory Root Fraction   9 =  0.20 0.40 0.40
Understory Root Fraction  9 =  0.40 0.60 0.00

############################ VEGETATION 10 ########################################
Vegetation Description    10 = Clear-cuts
Impervious Fraction       10 = 0.0
Detention Fraction        10 = 0
Detention Decay           10 = 0
Overstory Present         10 = FALSE
Understory Present        10 = TRUE
Fractional Coverage       10 =
Hemi Fract Coverage       10 =
Clumping Factor           10 =
Leaf Angle A              10 =
Leaf Angle B              10 =
Scattering Parameter      10 =
Trunk Space               10 =
Aerodynamic Attenuation   10 =
Radiation Attenuation     10 =
Max Snow Int Capacity     10 =
Snow Interception Eff     10 =
Mass Release Drip Ratio   10 =
Height                    10 = 0.5
Overstory Monthly LAI     10 =
Understory Monthly LAI    10 = 0.5 0.5 0.5 0.5 0.5 6.0 6.0 6.0 6.0 0.5 0.5 0.5
Maximum Resistance        10 = 600
Minimum Resistance        10 = 280
Moisture Threshold        10 = 0.33
Vapor Pressure Deficit    10 = 4000
Rpc                       10 = .108
Overstory Monthly Alb     10 =
Understory Monthly Alb    10 = 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19 0.19
Number of Root Zones      10 = 3
Root Zone Depths          10 = 0.10 0.25 0.40
```

```
Overstory Root Fraction  10 = 0.20 0.40 0.40
Understory Root Fraction 10 = 0.40 0.60 0.00

############################## VEGETATION 11 ##########################################
Vegetation Description   11 = Rock
Impervious Fraction      11 = 0.0
Detention Fraction       11 = 0
Detention Decay          11 = 0
Overstory Present        11 = FALSE
Understory Present       11 = FALSE
Fractional Coverage      11 =
Hemi Fract Coverage      11 =
Clumping Factor          11 =
Leaf Angle A             11 =
Leaf Angle B             11 =
Scattering Parameter     11 =
Trunk Space              11 =
Aerodynamic Attenuation  11 =
Radiation Attenuation    11 =
Max Snow Int Capacity    11 =
Snow Interception Eff    11 =
Mass Release Drip Ratio  11 =
Height                   11 =
Overstory Monthly LAI    11 =
Understory Monthly LAI   11 =
Maximum Resistance       11 =
Minimum Resistance       11 =
Moisture Threshold       11 =
Vapor Pressure Deficit   11 =
Rpc                      11 =
Overstory Monthly Alb    11 =
Understory Monthly Alb   11 =
Number of Root Zones     11 = 3
Root Zone Depths         11 = 0.10 0.25 0.40
Overstory Root Fraction  11 = 0.20 0.40 0.40
Understory Root Fraction 11 = 0.40 0.60 0.00

############################## VEGETATION 12 ##########################################
Vegetation Description   12 = Wetlands
Impervious Fraction      12 = 0.0
Detention Fraction       12 = 0
Detention Decay          12 = 0
Overstory Present        12 = FALSE
Understory Present       12 = TRUE
Fractional Coverage      12 =
Hemi Fract Coverage      12 =
Clumping Factor          12 =
Leaf Angle A             12 =
Leaf Angle B             12 =
Scattering Parameter     12 =
Trunk Space              12 =
Aerodynamic Attenuation  12 =
Radiation Attenuation    12 =
Max Snow Int Capacity    12 =
Snow Interception Eff    12 =
```

```
Mass Release Drip Ratio  12 =
Height                   12 = 0.5
Overstory Monthly LAI    12 =
Understory Monthly LAI   12 = 0.5 0.5 0.5 0.5 0.5 6.0 6.0 6.0 6.0 0.5 0.5 0.5
Maximum Resistance       12 = 600
Minimum Resistance       12 = 200
Moisture Threshold       12 = 0.33
Vapor Pressure Deficit   12 = 4000
Rpc                      12 = 0.108
Overstory Monthly Alb    12 =
Understory Monthly Alb   12 = 0.19 0.19 0.19 0.17 0.17 0.16 0.16 0.16 0.16 0.17 0.19 0.19
Number of Root Zones     12 = 3
Root Zone Depths         12 = 0.10 0.25 0.40
Overstory Root Fraction  12 = 0.20 0.40 0.40
Understory Root Fraction 12 = 0.4 0.6 0.0

############################ VEGETATION 13 ########################################
Vegetation Description   13 = Shoreline
Impervious Fraction      13 = 0.0
Detention Fraction       13 = 0
Detention Decay          13 = 0
Overstory Present        13 = FALSE
Understory Present       13 = TRUE
Fractional Coverage      13 =
Hemi Fract Coverage      13 =
Clumping Factor          13 =
Leaf Angle A             13 =
Leaf Angle B             13 =
Scattering Parameter     13 =
Trunk Space              13 =
Aerodynamic Attenuation  13 =
Radiation Attenuation    13 =
Max Snow Int Capacity    13 =
Snow Interception Eff    13 =
Mass Release Drip Ratio  13 =
Height                   13 = 0.5
Overstory Monthly LAI    13 =
Understory Monthly LAI   13 = 0.5 0.5 0.5 0.5 0.5 6.0 6.0 6.0 6.0 0.5 0.5 0.5
Maximum Resistance       13 = 600
Minimum Resistance       13 = 200
Moisture Threshold       13 = 0.33
Vapor Pressure Deficit   13 = 4000
Rpc                      13 = .108
Overstory Monthly Alb    13 =
Understory Monthly Alb   13 = 0.19 0.19 0.19 0.17 0.17 0.16 0.16 0.16 0.16 0.17 0.19 0.19
Number of Root Zones     13 = 3
Root Zone Depths         13 = 0.10 0.25 0.40
Overstory Root Fraction  13 = 0.20 0.40 0.40
Understory Root Fraction 13 = 0.40 0.60 0.00

############################ VEGETATION 14 ########################################
Vegetation Description   14 = Water
Impervious Fraction      14 = 0.0
Detention Fraction       14 = 0.0
Detention Decay          14 = 0.0
```

```
Overstory Present          14 = FALSE
Understory Present         14 = FALSE
Fractional Coverage        14 =
Hemi Fract Coverage        14 =
Clumping Factor            13 =
Leaf Angle A               13 =
Leaf Angle B               13 =
Scattering Parameter       13 =
Trunk Space                14 =
Aerodynamic Attenuation    14 =
Radiation Attenuation      14 =
Max Snow Int Capacity      14 =
Snow Interception Eff      14 =
Mass Release Drip Ratio    14 =
Height                     14 =
Overstory Monthly LAI      14 = 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Understory Monthly LAI     14 = 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Maximum Resistance         14 =
Minimum Resistance         14 =
Moisture Threshold         14 =
Vapor Pressure Deficit     14 =
Rpc                        14 =
Overstory Monthly Alb      14 = 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
Understory Monthly Alb     14 = 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
Number of Root Zones       14 = 3
Root Zone Depths           14 = 0.10 0.25 0.40
Overstory Root Fraction    14 = 0.20 0.40 0.40
Understory Root Fraction   14 = 0.00 0.00 0.00

##############################################################################
# MODEL OUTPUT SECTION
##############################################################################
[OUTPUT]                                  # Information what to output when
Output Directory  = ../output/
Initial State Directory   = ../modelstate/

################## PIXEL DUMPS ##########################################################

Number of Output Pixels    = 0

# For each pixel make a key-entry pair as indicated below, varying the
# number for the output pixel  (1, .. , Number of Output Pixel)

North Coordinate           1 =
East Coordinate            1 =
Name                       1 =

################## MODEL STATE ##########################################################
Number of Model States     =  0           # Number of model states to dump

# For each model state make a key-entry pair as indicated below, varying the
# number for the model state dump (1, .. , Number of Model States)

State Date                 1 = 10/02/1997-03
```

```
################ MODEL MAPS ##################################################

Number of Map Variables     = 0             # Number of different variables for
                                            # which you want to output maps

# For each of the variables make a block like the one that follows, varying
# the number of the variable (n = 1, .. , Number of Map Variables)

Map Variable              1 = 503           # water table depth
Map Layer                 1 = 1
Number of Maps            1 = 9
Map Date 1                1 = 12/8/1997-03
Map Date 2                1 = 12/8/1997-06
Map Date 3                1 = 12/8/1997-09
Map Date 4                1 = 12/8/1997-12
Map Date 5                1 = 12/8/1997-18
Map Date 7                1 = 12/8/1997-15
Map Date 6                1 = 12/8/1997-21
Map Date 8                1 = 12/9/1997-03
Map Date 9                1 = 12/9/1997-06
################ MODEL IMAGES ################################################

Number of Image Variables  = 0              # Number of variables for which you
                                            # would like to output images

# For each of the variables make a block like the one that follows, varying
# the number of the variable (n = 1, .. , Number of Image Variables)

Image Variable          1 =                  # ID of the variable to output
Image Layer             1 = 1                # If the variable exists for a number
                                             # of layers, specify the layers here
                                             # with the top layer = 1
Image Start             1 =                  # First timestep for which to output
                                             # an image
Image End               1 =                  # Last timestep for which to output
                                             # an image
Image Interval          1 =                  # Time interval between images (hours)
Image Upper Limit       1 =                  # All values in the output equal to or
                                             # greater than this limit will be set
                                             # to 255
Image Lower Limit       1 =                  # All values in the output equal to or
                                             # smaller than this limit will be set
                                             # to 0
################ GRAPHIC IMAGES ##############################################

Number of Graphics       =  0                # Number of variables for which you
                                             # would like to output images
Graphics ID             1 = 15               # ID of the variable to output
Graphics ID             1 = 22
Graphics ID             2 = 23
Graphics ID             3 = 24
Graphics ID             4 = 25
Graphics ID             5 = 43
Graphics ID             6 = 44
Graphics ID             7 = 8
```

```
Graphics ID             8 = 2
Graphics ID             9 = 50
Graphics ID             10 = 1

# 1      SWE (mm)
# 2      Water Table Depth (mm)
# 3      Digital Elevation Model (m)
# 4      Vegetation Class (index #)
# 5      Soil Class (index #)
# 6      Soil Depth (mm)
# 7      Precipitation at current time step (mm/time step)
# 8      Incoming Shortwave (Beam and Diffuse) (W/sqm)
# 9      Intercepted Snow (mm)
# 10     Snow Surface Temp (C)
# 11     Cold Content of snow entire snow pack (kJ)
# 12     Snow Melt (as Outflow minus Precip, can be negative) (mm/time step)
# 13     Snow Pack Outflow (mm/time step)
# 14     Saturated Subsurface Flow (mm/time step)
# 15     Overland Flow(mm)
# 16     Total Evapotranspiration (soil + all veg layers)
# 17     Ground Snow pack vapor flux (mm)
# 18     Intercepted snow pack vapor flux (mm)
# 19     Soil Moisture (Surface Layer) % of saturation (i.e. porosity)
# 20     Soil Moisture (2nd Layer) % of saturation (i.e. porosity)
# 21     Soil Moisture (3rd Layer) % of saturation (i.e. porosity)
# 22     Accumulated Precip (mm)
# 23     air temperature (C)
# 24     wind speed (m/s)
# 25     relative humidity
# 26     Prism Precip Field (mm)
# 31     Overstory Transpiration (mm)
# 32     Understory Transpirtation (mm)
# 33     Soil Evaporation (mm)
# 34     Overstory Evaporation (mm)
# 35     Understory Evaportation (mm)
# 41     Sky View Factor (%)
# 42     Shade Map (%)
# 43     Direct Beam Shortwave Rad (W/sqm)
# 44     Diffuse Beam Shortwave Rad (W/sqm)
# 45     Aspect (degrees)
# 46     Slope (percent)
# 50     Channel Subsurface Interception (mm)
# 51     Road Subsurface Interception (mm)
# WARNING Use soil mositure layers with caution, to minimize calculations during redraw
# DHSVM does not check to make sure that the assigned soil layer exists

################################################################################
# END OF INPUT FILE
################################################################################
[End]                                   # This is probably not needed, but
                                        # just in case (to close the previous
                                        # section)
```