

University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

JFSP Research Project Reports

U.S. Joint Fire Science Program

2010

A Proposal to Improve Performance of the Forest Vegetation Simulator - Fire and Fuels Extension

Dave C. Cawrse

US Forest Service

Michael G. Van Dyke

US Forest Service

Nicolas Nicholas L. Crookston

US Forest Service


Donald Robinson

ESSA Technologies Ltd

Sarah Beukema

ESSA Technologies Ltd

Follow this and additional works at: <http://digitalcommons.unl.edu/jfspresearch>

 Part of the [Forest Biology Commons](#), [Forest Management Commons](#), [Natural Resources and Conservation Commons](#), [Natural Resources Management and Policy Commons](#), [Other Environmental Sciences Commons](#), [Other Forestry and Forest Sciences Commons](#), [Sustainability Commons](#), and the [Wood Science and Pulp, Paper Technology Commons](#)

Cawrse, Dave C.; Van Dyke, Michael G.; Crookston, Nicolas Nicholas L.; Robinson, Donald; and Beukema, Sarah, "A Proposal to Improve Performance of the Forest Vegetation Simulator - Fire and Fuels Extension" (2010). *JFSP Research Project Reports*. 19.

<http://digitalcommons.unl.edu/jfspresearch/19>

This Article is brought to you for free and open access by the U.S. Joint Fire Science Program at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in JFSP Research Project Reports by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

A Proposal to Improve Performance of the Forest Vegetation Simulator - Fire and Fuels Extension

JFSP Project ID: 10-S-02-4

Principal Investigator:

Dave C. Cawrse, US Forest Service, Washington Office, Forest Management Service Center, Fort Collins, CO

Co-Principal Investigator:

Michael G. Van Dyck, US Forest Service, Washington Office, Forest Management Service Center, Fort Collins, CO

Contributing Investigators:

Nicolas Nicholas L. Crookston, US Forest Service, Rocky Mountain Research Station, Forestry Sciences Laboratory, Moscow, ID

Donald Robinson, ESSA Technologies Ltd, Vancouver, British Columbia, Canada

Sarah Beukema, ESSA Technologies Ltd, Vancouver, British Columbia, Canada

This research was sponsored in part by the Joint Fire Science Program. For further information go to www.firescience.gov



I. Abstract

The Forest Vegetation Simulator (FVS) and its associated Fire and Fuels Extension (FFE) have been used to provide information required by larger software systems like the Interagency Fuels Treatment - Decision Support System (IFT-DSS). Interacting with FVS in an automated fashion has been difficult, and simulations with very large numbers of stands, such as those necessary for landscape analyses for fire planning, could take a significant amount of time to process. This project was designed to: (A) develop a requirements document considering Service Oriented Architecture and how that may apply to FVS, and how FVS will be used interactively; (B) profile the FVS code to evaluate what takes the most processing time and identify possible areas for program optimization; (C) while optimizing and reducing the size of code, migrate FVS to a modern development framework such as Intel Fortran and the Visual Studio IDE; (D) identify platforms and systems that meet needs of the JFSP and other stakeholders, such as creating dynamic link libraries (DLL); and (E) specify and define the use of new technologies in the next phase of software development, such as OpenMP directives, thus implementing multithreading in the base FVS executables or extensions to take advantage of increased computing power of multicore processors.

II. Background and Purpose

Over the past four decades the Forest Vegetation Simulator has proven to have a well-designed, modular and extensible architecture for simulating forest growth. It has been successfully migrated from punch cards and mainframes to software development environments and PCs, jumping across a number of operating systems in the process.

In recent years needs have been identified to further improve the efficiency of FVS; where possible making use of multi-core hardware, and finding ways to allow it to be more dynamically embedded in other software systems. These things provided the impetus for this project.

III. Study Description and Location

At the outset of this project it was possible to run FVS programmatically through scripts, but it was not possible to interact fully with FVS during run execution. Implementation of a stop-start mechanism was explored to provide that capability. Source code was analyzed to find routines that could be modified to improve processing efficiency and speed, and the identified code was modified. Additional changes to the FVS source code were explored to make it possible for a supervisory program to interact with FVS during a simulation, fetching information about the current stand structure, dynamically changing management directives and stand structure while a simulation is taking place.

Source code changes were required to divide FVS into modules (DLLs in Windows versions; shared libraries in Unix operating systems) to simplify the update process and help reduce the memory and maintenance footprint in the future. The FVS debugging and testing environment needed to be widened to include a variant-wide build system for Windows that includes Visual Studio 2010 and a system independent build package using

the Cmake package, replacing the existing Windows-makefile system and providing a single build system that works across multiple operating systems.

In order to facilitate all these changes an open-source code and utility repository was developed. This open development environment is called open-fvs and is based on the Subversion (SVN) version control system hosted by the Google code project at <http://code.google.com/p/open-fvs/>.

IV. Key Findings

Contained herein are discussions of key findings related to each of the stated objectives, namely (A) developing a requirements document considering Service Oriented Architecture and how that may apply to FVS, and how FVS will be used interactively; (B) profiling the FVS code to evaluate what takes the most processing time and identify possible areas for program optimization; (C) while optimizing and reducing the size of code, migrating FVS to a modern development framework such as Intel Fortran and the Visual Studio IDE; (D) identifying platforms and systems that meet needs of the JFSP and other stakeholders, such as creating dynamic link libraries (DLL); and (E) specifying and defining the use of new technologies in the next phase of software development, such as OpenMP directives, thus implementing multithreading in the base FVS executables or extensions to take advantage of increased computing power of multicore processors.

A. Requirements document

A meeting was held in November of 2010 to discuss what would be needed to accomplish the goals of the project. Specific discussion topics included (1) the functional requirements of the software, including growth, mortality, fuels, volume, and management; (2) model speed; (3) release version run times; (4) MS Office compatibility; (5) modern development platforms; and (6) an outline to proceed with the work. Results of the discussions were used to develop a requirements document.

B. Profiling the FVS code

FVS execution speed was identified at project scoping as a strong candidate for improvement. Scenarios listed in Table 1 were used to evaluate execution performance. Note the scenario designated as Base includes the Fire and Fuels Extension (FFE) and the Stand Visualization System (SVS) file creation.

Table 1 - Scenarios used to study FVS execution performance

Scenario	Notes
Base	FFE active, SVS active
Base – FM	FFE references removed; SVS active
Base – FM – SV	FFE and SVS references removed

A 30-stand FVS-CR simulation of each scenario was profiled using AMD CodeAnalyst. The results shown in Figure 1 show the absolute time spent in each of ten functional

categories; those in Figure 2 show the relative time by scenario. What is clear from a comparison of the Base and Base-FM scenarios is the very high computational cost imposed by the FFE, and that most of this cost is due to volume calculations. In the Base scenario, about 45% of the time is spent in volume-functions. Moreover, about 90% of internal Math function calls are also related to volume calculations.

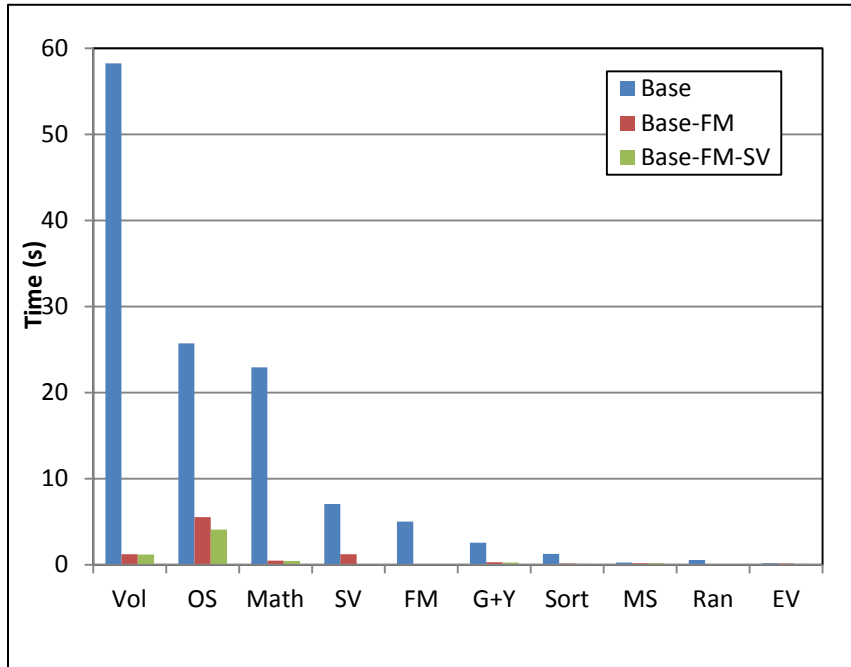


Figure 1 - Absolute time spent in each category of FVS subroutines. Vol = Volume; OS = Operating System; Math = Math functions; SV = Stand Visualization System; FM = Fire Model; G+Y = Growth & Yield; Sort = Sorting; MS = Dwarf Mistletoe; Ran = Random number; EV = Event Monitor. OS and Math categories are not accessible to the programmer.

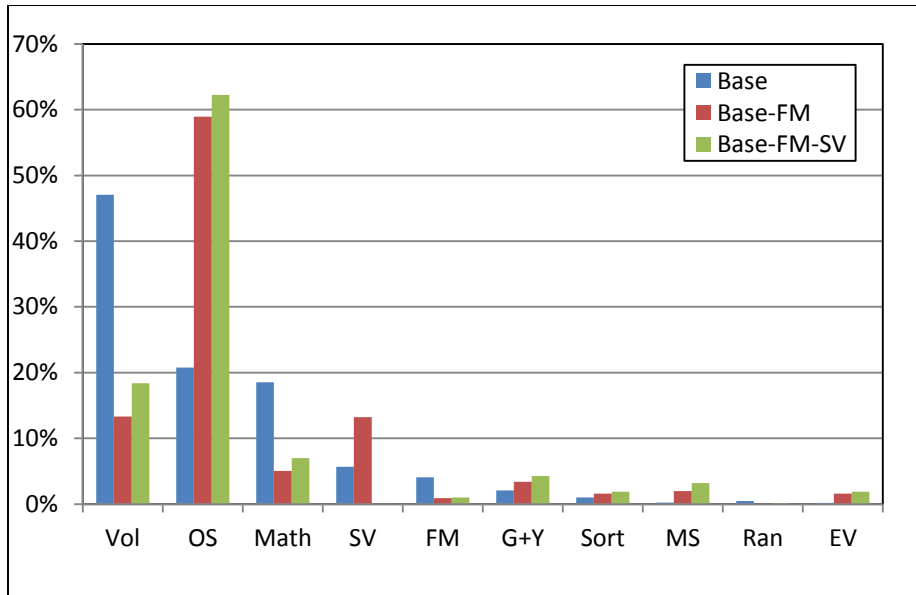


Figure 2 - Relative time spent in each category of FVS subroutines, rescaled by scenario. Abbreviations: see Figure 1.

C. Code optimization and modern development framework

This item is actually two significant items. The first is code optimization to increase the processing speed of the model. The second is migration to a more modern development framework.

1. Code optimization

Because of the extremely high computational cost of the FFE volume calculations, this was the focus of the code modifications related to processing optimization. The high cost was caused by the annual recalculation of live tree volume using the volume subroutines. In the case of snags, volume calculations were made twice each year to account for height loss; again using volume subroutines. Based on consultation with the FVS staff (Stephanie Rebain, pers. comm.), volume calculations were greatly simplified, and reporting was reduced to every FVS time-step rather than annually.

2. Modern development framework

FVS executables had traditionally been built using a make system that relied upon a DOS-based compiler with limited potential for modernization. Microsoft Visual Studio 2010 and Intel Visual Fortran 2011 were selected as the target development framework. Use of these tools, however, required the use of a new make system as well. Cmake was chosen. It is a powerful, well-supported open source software system for building-debugging-deploying software across multiple operating systems. A single master instruction file suffices for the creation of build systems targeting specific compilers and operating systems. When run on a Windows system with Visual Studio 2010 and Intel Fortran already installed, running this from the command line requires a single command, and automatically creates Solution Files (SLNs) for each FVS variant, requiring only a

listing of the source code required for each variant. In addition to its compatibility with Visual Studio, the Cmake system also creates project files and outputs for the MinGW system under Windows, as well as the makefile system under Unix. Source code listings have been completed for every variant. Each variant SLN can then be compiled in Debug or Release mode through the Visual Studio UI and then distributed to FVS users through the usual distribution channels. In addition, a batch compile utility is available so that all FVS variants can be compiled automatically. Users with the software can also compile and use customized or experimental versions of FVS that they have created themselves.

When the code base is linked to a repository system (either open-FVS or the Fort Collins repository), then code changes can be automatically applied to create updated Debug and Release executables.

D. Systems that meet the needs of stakeholders

In order to meet the needs of JFSP for systems like IFT-DSS, significant changes were required in the FVS source code. These can be grouped into two main categories: programmatic interaction and dynamic linking.

1. Programmatic interaction

FVS has traditionally been run by entering the program name at the system command line and then answering prompts for the names of input and output file names. A command line option has been added that replaces the prompts and allows the user to specify the file names as part of the command.

The command line capability allowed for development of a new facility that causes the program to stop during a simulation. FVS creates a stop point file and writes to it the state of all the stands at the specified point in time and at the specified point in the simulation computations. The simulation can then be restarted, exactly recovering the state variables from the specified stop point file and continuing processing from where it left off. This stop/restart facility allows for the precise interaction of FVS with supervisory control programs like IFT-DSS, including changes to stand structure and management.

A set of Application Program Interface (API) routines were developed for FORTRAN, VB.Net, and R applications. These routines allow users to interact programmatically with FVS. These APIs are documented at http://code.google.com/p/open-fvs/wiki/FVS_API, and are intended to be extensible and as needs are defined and refined, each can be elaborated.

The FVS program is run by calling a single API function, which returns when the simulation is over unless the program is requested to stop during the simulation. The request to stop is done by setting a stop point and stop year. The program returns when it reaches the designated stop point and then it picks up the calculations were it left off when it is called again. There are several ways to interact with the program between its "return" and when it is called again; each supported with an API function.

API functions have been developed for a wide variety of uses. For example, stopping and restarting are done through API functions. Information can be requested for summary

statistics, tree attributes, and event monitor variables. Trees can be added, harvested, and planted. Growth and mortality can be modified to simulate between-stand effects in landscape simulations, or to accommodate new growth and mortality models provided by the supervisory program. The limits for FVS code arrays can even be requested.

2. Dynamic linking

DLLs (Dynamic Link Libraries) are Windows files that provide compiled code that can be used by multiple programs. DLLs serve a similar role to source code library (LIB) files, but whereas the subroutines and functions in LIBs are incorporated at link time into an executable file (EXE), DLLs are not incorporated and are loaded “more or less” only when the EXE is run. This makes it possible to swap DLLs with updated versions without recompiling, and also to share common functionality across different executables.

To enable DLLs in versions compiled for Windows with VS2010, small language-specific changes have been made to the code so that: (a) arguments can be correctly passed between languages (two of the FVS DLL components used by all variants are written in the C language, and a third is written in the C++ language); and (b) subroutines found in the DLLs can be properly tracked in temporary “export” libraries prior to their references being incorporated into the final EXE, when all the components are linked together. At run time this allows FVS to identify where to find the subroutines referred to in different parts of the program.

Beginning about 2003, SQL connectivity was introduced through the Database (DBS) extension, which allowed input and output to be located in Excel or Access files, as well as SQL databases. This required the use of third party components built for the 32-bit ODBC protocol, purchased from Canaima Software. This company ceased operation around 2006, making it impossible to compile FVS for 64-bit Windows systems. Intrinsic ODBC functions were rewritten, entirely removing the need for Canaima software and opening the door to 64-bit executables. The FVSql DLL provides 23 interfaces between intrinsic SQL-ODBC functions and pre-existing FORTRAN equivalents found in the DBS extension. Using the mkdbstypeDefs project provided with the Cmake system, an OS-dependent file called TYPEDEFS.F77 is created, which provides correct declarations for both 32- and 64-bit builds.

When FVS was first created some software design standards differed from those in common use today. In the case of FVS, simulations with very serious errors were terminated by calling a FORTRAN STOP command.

Modern programs typically terminate by backing out through the subroutine call tree, returning control to the program’s starting location (usually called MAIN). Following this modern practice is mandatory for implementations of FVS that are embedded within a supervisory program written in a programming environment such as R, VB.Net or IFT-DSS. Otherwise, the supervisory program will “hang” when the FVS DLL ends through a FORTRAN STOP. This requirement has been met by modifying the error-handling routines so that when a serious error occurs, program control returns to the main entry point into the FVS DLL, via the API call to FVS subroutine. This modification handles most abnormal program terminations, but will not deal with program faults such as those caused by results of calculations such as LOG(-1).

E. New technologies

At the outset of this project we anticipated that multi-core parallelization would provide efficiency gains for FVS simulations. Since most personal computers now have 4 CPU cores with high performance systems having many more, it seemed reasonable to expect that dividing up the simulation to use more cores would provide good gains in processing speed. However, after the project scoping meeting it became clear that the most intensive FVS tasks are multi-stand simulations, and that the simplest implementation of a multi-core functionality would divide FVS processing tasks at the stand level, and not at the finer tree level, such as might be done for volume calculations, which are known to be compute-intensive.

We now believe that code to take advantage of multi-core computers is best implemented in supervisory programs which control multi-stand FVS simulations. This will simplify the implementation of directives such as OpenMP, assigning the simulation of each stand or polygon based on the available pool of CPUs.

A new technology that was implemented during the project but which was not anticipated at the outset is the use of an open-source repository system. Prior to this project the FVS source code was housed in an internal repository accessible only from inside the Forest Service firewall. A Google Code repository called open-fvs was set up to make the source code and related files available so that university, private, other federal, and state organizations who wish to participate in enhancing FVS can do so without the impediments caused by not having access to the Forest Service's internal repository.

A second new technology that was not foreseen at the start of the project is the Cmake system, which, allows all FVS to be built and tested using 3 tool chains (Visual Studio/Intel, MinGW and Unix), potentially including 32- and 64-bit versions.

V. Management Implications

Interacting with FVS in an automated fashion has traditionally been difficult. For example, the information required by larger software systems like the Interagency Fuels Treatment - Decision Support System (IFT-DSS) required running FVS simulations through to completion, and then obtaining the necessary information from an output file. In addition, simulations with very large numbers of stands, such as those necessary for landscape analyses for fire planning, could take a significant amount of time to process due to code inefficiencies.

The creation of a dynamic link library (DLL) in addition to the standard executable will allow the functionality of FVS to be more easily integrated into larger software systems. The protocols associated with calling a DLL and exchanging information with it are now well established and are clearly documented in the open-fvs wiki.

Allowing for stopping and restarting a simulation will allow for the exchange of information during a simulation instead of having to wait until a simulation has run to completion. This is important for systems that control the time element of a simulation and need to do other things while the FVS simulation waits to continue processing.

Programmatic interaction with the simulation in conjunction with the stopping/restarting capability allows for much more robust integration of FVS into larger software systems than has ever been possible before. A supervisory system can repeatedly stop the FVS simulation, retrieve information, use that information for whatever purpose is required, pass information back to FVS, and then command FVS to continue the simulation.

Improving processing efficiency and speed has the obvious benefit of decreasing the processing time for simulations with very large numbers of stands. This will help in situations where FVS was causing significant delays when included as a part of a larger software system.

VI. Ongoing Work on This Topic

The FVS source code is continuously being modified and enhanced. The effort to improve processing efficiency and speed has continued, and several potential areas for improvement have been identified.

Although the completed API routines satisfy the requirements for the deliverables of this project, development and refinement of API routines continues.

VII. Future Work Needed

The open-fvs open source code repository will allow for a greater degree of collaboration with additional developers than has ever been possible in the past. Realizing the full potential of the FVS model will require collaboration with people in different disciplines from across the country and beyond and include the next generation of model developers and mensurationists.

There are certainly opportunities for continued improvements in the interactivity with other software. Systems like the Rangeland Vegetation Simulator, which rely on frequent interaction with FVS, will benefit from these improvements. Interaction with a supervisory program will allow for multi-stand simulation with stand-to-stand interaction, as has been demonstrated using R as the supervisor. A supervisory program would also allow for creation of a raster-based system, as opposed to the traditional stand-based system.

There are opportunities for continued improvements in processing efficiency, speed, and stability, particularly with the addition of fully tested 64-bit tool chains to complement the existing 32-bit chains, working examples of multi-core supervisory code, and improved error handling for low level mathematical errors.

Further enhancement of the API is straightforward, and could include the addition of more FFE variables (e.g. fuel models, coarse woody debris pool) and automatic on-demand metric conversion (currently begun in VB.Net). As new features are developed for the FVS model these improvements will all become increasingly important.

VIII. Deliverables Crosswalk Table

Deliverable	Status
Profile the source code to evaluate processing time	Completed
Specify performance standards	Completed
Conduct a planning workshop to assess needs	Completed
Develop a requirements document	Completed
Migrate FVS to the Intel / Visual Studio development environment	Completed
Develop efficient code - optimize routines, delete outdated code	Completed
Develop DLLs for three FVS variants	Completed
As needed, modify FVS code to be multi-threaded	Not needed