Computers and Electronics in Agriculture 81 (2012) 62–71

Contents lists available at SciVerse ScienceDirect

Computers and Electronics in Agriculture

journal homepage: www.elsevier.com/locate/compag

# Methodology to evaluate the performance of simulation models for alternative compiler and operating system configurations ☆

K.R. Thorp [a,*], J.W. White [a], C.H. Porter [b], G. Hoogenboom [c], G.S. Nearing [d], A.N. French [a]

[a] USDA-ARS, US Arid Land Agricultural Research Center, 21881 N Cardon Ln, Maricopa, AZ 85138, United States
[b] University of Florida, Department of Agricultural and Biological Engineering, 243 Rogers, Gainesville, FL 32611, United States
[c] Washington State University, AgWeatherNet, 162 Hamilton Hall, Prosser, WA 99350, United States
[d] University of Arizona, Department of Hydrology and Water Resources, Harshbarger Building, Tucson, AZ 85721, United States

ARTICLE INFO

*Article history:*
Received 7 May 2011
Received in revised form 30 September 2011
Accepted 23 November 2011

*Keywords:*
Crop model
G95
Gfortran
Linux
Open source
Windows

ABSTRACT

Simulation modelers increasingly require greater flexibility for model implementation on diverse operating systems, and they demand high computational speed for efficient iterative simulations. Additionally, model users may differ in preference for proprietary versus open-source software environments. These issues necessitate the development of strategies to maximize model compatibility across operating systems, to ensure numerically accurate simulations for alternative compiler selections, and to understand how these choices affect computational speed. We developed an approach to evaluate model performance using diverse Fortran compilers on multiple computer operating systems. A single desktop computer with five identical hard drives was designed to permit meaningful comparisons between five operating systems while minimizing differences in hardware configuration. Three Fortran compilers and relevant software development tools were installed on each operating system. Both proprietary and open-source versions of compilers and operating systems were used. Compatibility and performance issues among compiler and operating system combinations were assessed for an example case: the Cropping System Model (CSM) as implemented in version 4.5 of the Decision Support System for Agrotechnology Transfer (DSSAT). A simulation study that included 773 simulations and assessed the full suite of crop growth modules within DSSAT-CSM was conducted for each compiler and operating system configuration. For a given simulation, results were identical for anthesis date (ADAT), maturity date (MDAT), and maximum leaf area index (LAIX) regardless of the compiler or operating system used. Over 94% of the simulations were identical for canopy weight at maturity (CWAM) and cumulative evapotranspiration at maturity (ETCM). Differences in CWAM were predominantly less than 2 kg $ha^{-1}$ and were likely the result of differences in floating point handling among compilers. Larger CWAM discrepancies highlighted areas for improvement of the model code. Model implementations with the Intel Fortran compiler on the Linux Ubuntu operating system provided the fastest simulations, which averaged 9.0 simulations $s^{-1}$. Evaluating simulation models for alternative compiler and operating system configurations is invaluable for understanding model performance constraints and for improving model robustness, portability, usefulness, and flexibility.

Published by Elsevier B.V.

## 1. Introduction

With recent advancements in computational power and capability, computer simulation models are becoming more widely utilized for analysis of biophysical processes in a variety of research disciplines. However, the applicability of a model is often limited to the computing environment on which it was developed and tested. This paper presents a methodology to test and compare simulation model performance, in terms of numerical accuracy and computational speed, when implemented using a variety of source code compilers on multiple operating systems. There are several compelling reasons to develop such a methodology. Primarily, the methodology is useful for comparing model performance across a variety of computing environments, each having unique advantages for model implementation and use. Second, because various compilers (and compiler options) handle numerical processing differently, the methodology can identify numerical discrepancies between compilers, which can lead to source code improvements that strengthen model robustness and reliability. Third, even if numerical consistency is not a major problem, the methodology is

☆ Mention of trade names or commercial products in this publication is solely for the purpose of providing specific information and does not imply recommendation or endorsement by the United States Department of Agriculture.

* Corresponding author.
*E-mail address:* kelly.thorp@ars.usda.gov (K.R. Thorp).

0168-1699/$ - see front matter Published by Elsevier B.V.
doi:10.1016/j.compag.2011.11.008

useful for comparing computational speed among compiler and operating system combinations and identifying approaches for improving the computational efficiency of the model. Finally, the methodology can support efforts to develop models in an 'open-source' software development paradigm, which emphasizes not only the provision of model source code, but also its usability with a complete suite of programming tools that are often, themselves, open-source.

An appropriate computing environment for model implementation depends heavily on the specific modeling application. For example, because of its prevalence on modern desktop computers, the Microsoft Windows operating system (Microsoft Corporation, Redman, Washington) is appropriate for most routine modeling tasks that require only moderate numbers of simulations. However, many modeling applications, such as optimization problems (Veith et al., 2003), model uncertainty analyses (Nol et al., 2010), model parameter estimation (Braga and Jones, 2004), data assimilation routines (Quaife et al., 2008), and spatial modeling applications (Thorp et al., 2007), require iterative calculations that can take many hours of time on desktop systems. Therefore, efforts to understand options for increasing the computational speed of iterative simulations on desktop systems is both useful and informative. The most intensive simulation analyses, however, are likely to be conducted in a high performance computing cluster, such as those available on many university campuses. One website (http://www.top500.org/stats/list/37/osfam) reports that 456 of the 500 fastest supercomputers in the world use the Linux operating system, while only 6 run Windows. Thus, simulation models developed for use only on Windows can not be readily implemented on most of the world's fastest computers. Efforts to broaden a model's applicability across a range of computing environments, specifically Linux, will likewise broaden its versatility, usefulness, and flexibility to function in a manner best suited for the modeling application at hand.

With improvements in computing technology over the past few decades, scientists are spending more time writing custom software. However, a recent survey of 2000 researchers showed that only 47% understood software testing protocols and only 34% thought that formal training in software development was important (Merali, 2010). Even corporate software giants have been criticized, as Knusel (2005) lamented the persistent numerical inaccuracies in statistical computations of Microsoft Excel. Similarly, Keeling and Pavur (2007) compared nine software packages for accuracy of statistical computations. Newer versions of the tested software demonstrated substantial improvements as compared to older versions, but they still found many areas where commonly used statistical packages could be improved. They concluded that software testing studies resulted in improved numerical robustness and reliability of statistical software. The same is very likely true for simulation models, many of which have been developed by scientists and graduate students with little formal software development training. The methodology presented herein provides a software testing environment to compare the effects of source code compilers and compiler settings on the numerical output of simulation models. Because compilers are inherently different, we propose a model development strategy that aims to minimize numerical differences among different compilations of the same model source code. Such efforts will broaden the model's versatility and reliability within a wider range of programming environments, which is an important characteristic for model development in the open-source paradigm.

As compared to traditional proprietary software development, the open-source software phenomenon has radically altered how software is developed and distributed (Hauge et al., 2010). Key differences include the adoption of software licenses that freely provide source code to end users and the establishment of developer communities whose members freely and collectively contribute to the project. Mockus et al. (2002) hypothesized that successful open-source software projects, such as Mozilla, have lower defect density than commercial software, because several testing teams have been established to maintain test cases and report defects. They also hypothesized that a lack of resources devoted to finding and repairing defects will ultimately lead to project failure. The open-source paradigm offers great opportunity for future development of simulation models, yet challenges and pitfalls clearly abound. Systematic testing procedures, such as the methodology presented herein, can facilitate open-source software development by identifying defects and other areas for coding improvement and by insuring model compatibility across a range of computing and programming environments.

The Fortran code of the Cropping System Model (CSM), currently packaged with the Decision Support System for Agrotechnology Transfer (DSSAT), has been developed over several decades by many agricultural scientists and their students (Jones et al., 2003; Hoogenboom et al., 2010). Many of the developers are self-taught programmers with minimal formal software development training. Yet, the effort has resulted in a software product that is used globally to address complex scientific problems, such as global climate change, water and nutrient cycling, and risk assessments for crop production (Boote et al., 1996, 2010; Tsuji et al., 1998). Increasingly, applications of the DSSAT-CSM are computationally intensive and iterative in nature, requiring hours or days to complete the simulations. For example, iterative techniques have been necessary for evaluation of precision nitrogen fertilization strategies (Paz et al., 1999), for estimation of cultivar coefficients (Anothai et al., 2008), and for assimilation of remotely sensed leaf area index (Thorp et al., 2010). The DSSAT-CSM development team recently decided to migrate the development of the software to the constructs of an open-source software project (Hoogenboom et al., 2011). Such action will provide a formal protocol for international collaboration on DSSAT-CSM development, and it will formalize the copyright and licensing of DSSAT-CSM to provide free access to the science contained within. However, the developers must insure that the freedoms of going open-source do not impair the scientific robustness of the product. These characteristics make the DSSAT-CSM an excellent example case for development and demonstration of our proposed methodology.

Our objective was to develop a methodology to compare the performance of simulation models across a variety of programming and computing platforms. Specifically, the methodology was used to evaluate model performance for alternative configurations of source code compilers and operating systems. Comparisons of numerical accuracy and computational speed among various compiler and operating system combinations provided guidance for improvement of model source code and suggested model implementation options that increase simulation speed. The methodology was effectively demonstrated using the DSSAT-CSM as an example case. Although the results of the analysis are necessarily specific to the DSSAT-CSM, the methodology itself has broad applicability to a wide range of simulation models and other software tools used for agricultural and environmental applications.

## 2. Materials and methods

### 2.1. The DSSAT-CSM example

The Cropping System Model (ver. 4.5.1.005) is a set of Fortran code that programatically synthesizes current knowledge of agroecosystem functionality. The model utilizes mass balance principles to simulate the carbon, nitrogen, and hydrologic processes and transformations that occur within agroecosystems. Simulations of

crop development and growth for over 25 crop species are possible. The CSM calculates agroecosystem processes within a homogeneous area on a daily time step, and certain subprocesses are computed hourly. Crop development proceeds through a series of growth stages based on photothermal time or heat unit accumulation from planting to harvest. Photosynthesis is computed using a radiation use efficiency approach in the CERES family of crop growth modules, including maize, wheat and other cereals. The CROPGRO family of crop modules, including soybean, peanut, dry bean, tomato, cotton, and many other crops, calculates leaf-level photosynthesis using a hedge-row approach for light interception. Assimilated carbon is partitioned to various plant parts, including leaves, stems, roots, and grain. Simulated plant growth responds to variation in management practices, cultivar selection, soil properties, and meteorological conditions. Management inputs required for model execution include plant population, row spacing, seed depth, planting dates, fertilizer application amounts and dates, and irrigation application amounts and dates. Cultivar parameters define day length sensitivity, heat units needed to progress through growth stages, and growth potentials for specific plant parts. Soils are defined by their water retention and conductivity characteristics, bulk density, pH, and initial conditions for water, inorganic nitrogen, and organic carbon. Daily inputs for minimum and maximum temperature and solar radiation are required for Priestley–Taylor calculations of evapotranspiration. Additional measurements of dew point temperature and wind speed allow for evapotranspiration calculations using FAO-56 methods. An energy balance approach for evapotranspiration calculation is also available when using the CROPGRO implementation. The model simulates plant stress effects from deficit and excess soil water conditions and from deficit soil nitrogen conditions, which feedback on the daily plant growth simulation.

The CSM Fortran code is complex, having 185,611 lines in 323 files. It is usually compiled with the Intel Fortran compiler (Intel Corporation, Santa Clara, CA) on a Microsoft Windows operating system (Microsoft Corporation, Redman, Washington). It is distributed with the Decision Support System for Agrotechnology Transfer (DSSAT) software (Hoogenboom et al., 2010), an agricultural decision support system designed for use on Microsoft Windows. DSSAT includes multiple software tools for creating input files for the CSM simulation model, for creatively implementing the CSM to answer agroecological questions, and for viewing and analyzing output from the CSM. Most importantly for the present study, DSSAT includes a broad set of management, cultivar, soil, and weather input files, developed using data from field experiments conducted around the world, for running all the various components of the CSM. We used this set of input files to test CSM performance for alternative Fortran compiler and operating system configurations. We also used the CSM version that is currently distributed with DSSAT 4.5, which was compiled with Intel Fortran on a separate Microsoft Windows 7 machine, as the baseline for comparison against our locally compiled CSM versions.

### 2.2. Computer hardware

To permit rapid testing of multiple operating systems, a custom-built desktop computer was used to conduct the simulations for this study. The computer was equipped with a TYAN motherboard (Tomcat K8E S2865, TYAN Computer Corporation, Taipei, Taiwan) and the central processing unit (CPU) was a 2.01 GHz, dual-core AMD Opteron (170, Advanced Micro Devices, Inc., Sunnyvale, California). Although a dual-core processor was available, the DSSAT-CSM code is not optimized for multi-core processing. Any parallelization of model simulations is thus dependent on the compiler and its settings. Four GB of random-access memory (RAM) were installed in the system. A swappable hard drive rack allowed for rapid, manual switching among hard drives, so multiple operating systems could be tested on the same computer hardware. Five identical 320 GB, serial ATA, internal hard drives (Barracuda 7200, Seagate Technology LLC, Scotts Valley, California) were placed in the trays designed for the swappable rack.

### 2.3. Operating systems

Five operating systems were installed on the hard drives. We tested three versions of Microsoft Windows (Microsoft Corporation, Redman, Washington), including Windows XP Professional (Service Pack 3, 32-bit), Windows XP Professional x64 Edition (Service Pack 2, 64-bit), and Windows 7 Professional (64-bit). Two versions of Linux operating systems, Linux Ubuntu (the Karmic Koala, ver. 9.10, 64-bit) and Linux Fedora (Sulphur, Release 9, 64-bit), were also tested. All of these operating systems support multithreading with a dual-core processor.

### 2.4. Photran under Eclipse

The Eclipse software (Helios, ver. 3.6; www.eclipse.org) was used to manage the compilations of the CSM Fortran code. Eclipse is an open-source software development platform that supports open-source integrated development environments (IDE) for multiple programming languages, including Java, Fortran, C/C++, Python, and others. The strength of Eclipse lies in its plug-in system, which is used to extend its functionality. An added advantage for the present study was its availability for both Windows and Linux operating systems.

Photran (ver. 6.0.3; www.eclipse.org/photran) is an IDE extension that provides a source code editor and debugging interface for Fortran programming in Eclipse. Fortran 77, 90, 95, and 2003 code is supported. Photran utilizes makefile-based compilation, so a makefile must be supplied to compile and build programs. Photran invokes a 'make' utility, which subsequently invokes the compiler commands within the makefile. Photran does not supply the make utility or the Fortran compiler directly. This gives the programmer flexibility to work with various Fortran compilers under the same IDE. To work with the CSM Fortran code in the present study, we installed the Photran IDE under Eclipse on each of the five operating systems.

### 2.5. Fortran compilers

Several Fortran compilers were used to compile the CSM code in this study. On the two Linux operating systems, we installed the Intel Fortran compiler for Linux (Composer XE, ver. 2011.3.174, Intel Corporation, Santa Clara, CA), the open-source 'g95' Fortran compiler (ver. 0.92; www.g95.org), and the open-source 'gfortran' compiler within the GNU compiler collection (ver. 4.5.1; gcc.gnu.org). The GNU (GNU's Not Unix, www.gnu.org) software system has resulted from a mass collaborative software development endeavor, ongoing since 1983, to provide an open-source, Unix-like, computer operating system. The GNU compiler collection (GCC) is one software package resulting from this collaboration. The gfortran compiler within GCC resulted from a fork in development of the g95 compiler in 2003. The g95 compiler is not included in the GCC. The gfortran and g95 Fortran compilers have similar roots but have diverged significantly since the fork in development.

On the three Microsoft Windows operating systems, we installed the Intel Fortran compiler for Windows (Professional Edition, ver. 2011.1.127, Intel Corporation, Santa Clara, CA), the open-source 'gfortran' compiler distributed with MinGW (Minimalist GNU for Windows, ver. 20100909; www.mingw.org), and the open-source 'g95' compiler built for use with MinGW (www.g95.org). An installation of Microsoft Visual Studio 2008 (Microsoft Corporation,

Redman, Washington) was a prerequisite for installing the Intel Fortran compiler on the Windows operating systems. We installed Visual Studio, but still used Photran under Eclipse to manage the Intel compiler. To implement the g95 and gfortran compilers on Windows, we installed MinGW on each Windows operating system. MinGW contains several open-source GNU software components, including the GCC, the GNU make utility, and GNU binutils, for compiling code in multiple programming languages. Additionally, MinGW exploits the standard Microsoft system DLL files to provide the C-runtime and Windows application programming interface (API). Thus, MinGW provides a completely open-source programming tool set for development of native Microsoft Windows applications that do not depend on third party C-runtime DLLs.

### 2.6. Compiler options

Each Fortran compiler has many options for controlling how programs are assembled, which ultimately determines the performance of the program in terms of numerical accuracy and computational speed. Most of the options were unique to each compiler, although desired effects of particular options were theoretically equivalent. For example, an important consideration is how the program should handle floating point exceptions: what should the program do if there is an attempt to divide by zero or take the square root of a negative number? In the distribution version of the DSSAT-CSM, the '/fpe:0' compiler flag in Intel Fortran is used to halt processing in case of overflow, divide-by-zero, and invalid floating point exceptions. In case of underflow or denormal numbers, the result is set to zero. We attempted to set compiler flags for g95 and gfortran to mimic the effect of the '/fpe:0' compiler flag in Intel Fortran. For the primary simulation scenarios (Table 1), the three compilers were invoked within the makefiles using the following commands for Intel, g95, and gfortran, respectively:

- ifort -fpe0 -O2 -c -o (obj) (src)
- g95 -O2 -c -o (obj) (src)
- gfortran -fd-lines-as-comments -ffpe-trap=invalid,zero,overflow -O2 -c -o (obj) (src)

where (obj) is the name of the object file and (src) is the name of the source file. The Linux-style form of the Intel floating point handling flag ('-fpe0') was required even on the Windows operating systems, because Eclipse used the MinGW libraries to compile the model. For gfortran, the '-fd-lines-as-comments' flag instructed the compiler to treat debug lines ('D' in the first column) as commented lines, and the '-ffpe-trap' flag dictated the floating point exceptions on which the program should halt. With gfortran, options were set to halt processing on overflow, invalid operations, and division by zero. We did not identify an option for flushing denormal numbers and underflows to zero with the gfortran compiler. Instead of using compiler flags, floating point exceptions with the g95 compiler are handled at run-time using environment variables. We set the g95 environment variables such that the program would halt on overflow, invalid operations, and division by zero and would flush denormal numbers to zero. We explicitly set the optimization flag ('-O2') to be identical for each compiler, since the default optimization flag varied among them. Additional simulation scenarios (Table 2) tested the effects of setting alternate values for the optimization and floating point handling flags.

### 2.7. Simulation study

A simulation study was performed for the various combinations of operating system and Fortran compiler. For each tested combination, we ran all the sample scenarios that are distributed with DSSAT 4.5. A total of 773 simulations were conducted for each combination of compiler and operating system (Table 1). Batch files were created to run all the sample scenarios for each crop type (Table 3). After each set of batch simulations, we extracted from the file output the simulation results for anthesis date (ADAT), physiological maturity date (MDAT), and maximum leaf area index (LAIX). These outputs were used as indicators to address the numerical accuracy of crop phenology and leaf area development simulations. We also extracted the output for canopy weight at maturity (CWAM) and cumulative evapotranspiration at maturity (ETCM). These two outputs were used as indicators of numerical accuracy in the biomass growth and hydrologic components of the model. Evaluating CWAM instead of grain yield allowed us to compare across the full suite of DSSAT-CSM crop modules, some of which do not produce grain yield outputs. Evaluating ETCM assessed the water balance in the pathway most likely to have the largest water fluxes. Any differences in the water balance simulation would likely cause discrepancies in the nitrogen balance as well. Together, ADAT, MDAT, LAIX, CWAM, and ETCM provided a

**Table 1**
Summary of primary simulation scenarios to compare various Fortran compilers and operating systems.[a]

| Identifier | Compiler | Operating system |
|---|---|---|
| D45[b] -XP32 | Intel Fortran | Windows XP |
| D45-XP64 | Intel Fortran | Windows XP x64 Edition |
| D45-WIN7 | Intel Fortran | Windows 7 |
| INT-XP32 | Intel Fortran | Windows XP |
| INT-XP64 | Intel Fortran | Windows XP x64 Edition |
| INT-WIN7 | Intel Fortran | Windows 7 |
| INT-UBUN | Intel Fortran | Linux Ubuntu 9.10 |
| INT-FEDO | Intel Fortran | Linux Fedora 9 |
| G95-XP32 | g95 | Windows XP |
| G95-XP64 | g95 | Windows XP x64 Edition |
| G95-WIN7 | g95 | Windows 7 |
| G95-UBUN | g95 | Linux Ubuntu 9.10 |
| G95-FEDO | g95 | Linux Fedora 9 |
| GFT-XP32 | gfortran | Windows XP |
| GFT-XP64 | gfortran | Windows XP x64 Edition |
| GFT-WIN7 | gfortran | Windows 7 |
| GFT-UBUN | gfortran | Linux Ubuntu 9.10 |
| GFT-FEDO | gfortran | Linux Fedora 9 |

[a] All primary scenarios were implemented from the hard disk drive with antivirus disabled, strict floating point handling enabled, and a compiler optimization setting of '-O2.'
[b] Indicates the compiled model distributed with DSSAT 4.5.

**Table 2**
Summary of secondary simulation scenarios to test specific conditions, including antivirus status, floating point handling options, optimization settings, and usage of random-access memory (RAM) disks.[a]

| Identifier | Compiler | Operating system | Test condition |
|---|---|---|---|
| INT-XP32-AV | Intel Fortran | Windows XP | Antivirus on |
| INT-XP64-AV | Intel Fortran | Windows XP x64 Edition | Antivirus on |
| INT-XP32-FH | Intel Fortran | Windows XP | No '-fpe0' |
| INT-XP64-FH | Intel Fortran | Windows XP x64 Edition | No '-fpe0' |
| INT-WIN7-FH | Intel Fortran | Windows 7 | No '-fpe0' |
| INT-XP32-Od | Intel Fortran | Windows XP | '-Od' flag |
| INT-XP32-O1 | Intel Fortran | Windows XP | '-O1' flag |
| INT-XP32-O3 | Intel Fortran | Windows XP | '-O3' flag |
| G95-XP32-O0 | g95 | Windows XP | '-O0' flag |
| G95-XP32-O1 | g95 | Windows XP | '-O1' flag |
| G95-XP32-O3 | g95 | Windows XP | '-O3' flag |
| GFT-XP32-O0 | gfortran | Windows XP | '-O0' flag |
| GFT-XP32-O1 | gfortran | Windows XP | '-O1' flag |
| GFT-XP32-O3 | gfortran | Windows XP | '-O3' flag |
| INT-XP32-RD | Intel Fortran | Windows XP | RAM disk |
| G95-XP32-RD | g95 | Windows XP | RAM disk |
| GFT-XP32-RD | gfortran | Windows XP | RAM disk |

[a] Unless noted otherwise, scenarios were implemented from the hard disk with antivirus disabled, strict floating point handling enabled, and a compiler optimization setting of '-O2.'

**Table 3**
Number of simulations for each crop type included in the primary scenario simulations.[a]

| Crop | Count |
|---|---|
| Bahia grass | 23 |
| Barley | 13 |
| Brachiaria | 24 |
| Cabbage | 31 |
| Cassava | 16 |
| Chickpea | 36 |
| Cotton | 5 |
| Cowpea | 15 |
| Dry bean | 65 |
| Faba bean | 6 |
| Fallow | 12 |
| Green bean | 5 |
| Maize | 58 |
| Millet | 6 |
| Peanut | 63 |
| Pepper | 5 |
| Potato | 14 |
| Rice | 39 |
| Sorghum | 6 |
| Soybean | 255 |
| Sugar cane | 16 |
| Sweet corn | 10 |
| Tomato | 10 |
| Velvet bean | 12 |
| Wheat | 28 |
| Total | 773 |

[a] Definitions of primary scenarios are found in Table 1.

well-rounded assessment of numerical performance for the complete cropping system simulation. For each batch simulation, we used the computer clock to measure the simulation duration. This provided an estimate for the computational speed of the model for the alternative compiler and operating system configurations. These activities were managed using a Python script within the Eclipse IDE on each operating system.

All primary simulation scenarios were implemented from the hard disk drive with antivirus software disabled, strict floating point handling enabled, and a compiler optimization setting of '-O2.' Several secondary simulation scenarios were subsequently designed to test the effect of these characteristics on numerical accuracy and computational speed (Table 2). First, since antivirus software is known to require substantial computer resources and reduce performance of other programs, we conducted the simulations with an enabled antivirus program (Symantec Endpoint Protection, ver. 11.0.6005.562, Mountain View, California) using the model compiled with Intel Fortran on two of the Windows operating systems. Second, we included three scenarios using the Intel Fortran compiler without the '-fpe0' floating point handling flag. These simulations were conducted on the three Windows operating systems. Third, we tested the effect of compiler optimization settings with the Intel, g95, and gfortran compilers on the Windows XP system. Each compiler had four optimization levels from '-O0' to '-O3.' For Intel, the '-O0' optimization level was instead '-Od.' Optimization allows the compiler to alter the code to make it faster, although numerical calculations may be affected in subtle ways and the details of optimization implementation may differ among compilers. Fourth, we used RAM disk software (RAMDisk ver. 3.5.130, DATARAM, Princeton, NJ) to establish a 2.5 GB system drive in RAM on Windows XP. The model executable file, input files, and batch files were then copied to the RAM drive, and simulations were conducted to test the effect of RAM disks on computational performance. These additional scenarios provided further understanding of model performance as affected by factors known to influence numerical processing and computational speed.

In a final test, sequential simulations with the CSM-CROPGRO-Soybean model were used to assess the effect of simulation length (number of years) on computational speed. Since the model accesses the weather file information only periodically during sequential simulations, we expected different results depending on the simulation length. Thirty-five iterations of this test were conducted to assess variability in computational speed. These CSM-CROPGRO-Soybean simulations were also used to test the effect of full versus minimum file output on computational speed. We used this approach to understand differences in file system efficiency among the operating systems.

## 3. Results and discussion

### 3.1. Numerical accuracy

Among all of the primary scenarios, simulation results for ADAT, MDAT, and LAIX were identical regardless of the Fortran compiler or operating system used (results not shown). For CWAM and ETCM, 94% of the 773 total simulations were identical (Table 4). Differences for CWAM and ETCM were more frequent among different compilers, because these model outputs were obtained from daily integration of canopy weight gains and evapotranspiration losses. Differences in floating point handling among compilers sometimes resulted in different CWAM and ETCM values as the model accumulated the daily fluctuations in mass flows.

Various groups of configurations provided identical results among themselves. For example, all simulations with the DSSAT 4.5 distribution version of the model gave identical results for CWAM and ETCM regardless of the Windows operating system used. Simulations with the model compiled in-house using the Intel Fortran compiler for Windows were also identical to the DSSAT 4.5 distribution version, regardless of the Windows operating system. When comparing the Intel Fortran compiler on Windows versus Linux operating systems, 13 simulations of CWAM and 4 simulations of ETCM were different. This was likely due to the difference in Intel compiler versions that we obtained for Windows and Linux. Simulations with the g95-compiled model were identical among the three Windows operating systems; however, 76 simulations of CWAM and 8 simulations of ETCM were different as compared to the DSSAT 4.5 distribution version of the model. This highlighted differences in the way the g95 and Intel Fortran compilers were interpreting the code on the Windows operation system. In comparing results for the g95 compiler between Windows and Linux operating systems, there were 71 differences in CWAM and 7 differences in ETCM. However, results were identical for simulations with the g95-compiled model on the Linux Ubuntu and Linux Fedora operating systems. Similar to g95, simulations with the gfortran-compiled model were identical on the three Windows operating systems, but differences were noted in comparing gfortran on Linux versus gfortran on Windows. These differences are likely due to the fact that the both the gfortran and g95 compilers are built and distributed uniquely for Windows and Linux operating systems. Since we had to download separate installation files for g95 and gfortran depending on whether it was for a Linux or a Windows operating system, there was no way to ensure that the two versions of each compiler were in fact identical. On the two Linux operating systems, both the g95-compiled model and the gfortran-compiled model gave identical results for CWAM and ETCM. Nevertheless, there were still a few differences between the simulations on Linux and the simulations with the DSSAT 4.5 distribution version of the model on Windows.

Of all the model outputs evaluated, CWAM typically had the greatest number of differing results among the operating system and Fortran compiler configurations. For a majority of the

**Table 4**
Number of simulations (out of 773 total) having different values for canopy weight at maturity (CWAM; upper right section) and cumulative evapotranspiration at maturity (ETCM; lower left section) for the primary simulation scenarios.[a]

| | D45-XP32 | INT-XP32 | INT-UBUN | INT-FEDO | G95-XP32 | G95-UBUN | G95-FEDO | GFT-XP32 | GFT-UBUN | GFT-FEDO |
|---|---|---|---|---|---|---|---|---|---|---|
| D45-XP32[b] | – | 0 | 13 | 13 | 76 | 38 | 38 | 78 | 38 | 38 |
| INT-XP32[2] | 0 | – | 13 | 13 | 76 | 38 | 38 | 78 | 38 | 38 |
| INT-UBUN | 4 | 4 | – | 0 | 80 | 43 | 43 | 81 | 43 | 43 |
| INT-FEDO | 4 | 4 | 0 | – | 80 | 43 | 43 | 81 | 43 | 43 |
| G95-XP32[b] | 8 | 8 | 8 | 8 | – | 71 | 71 | 32 | 71 | 71 |
| G95-UBUN | 7 | 7 | 9 | 9 | 7 | – | 0 | 73 | 0 | 0 |
| G95-FEDO | 7 | 7 | 9 | 9 | 7 | 0 | – | 73 | 0 | 0 |
| GFT-XP32[b] | 7 | 7 | 9 | 9 | 2 | 6 | 6 | – | 73 | 73 |
| GFT-UBUN | 7 | 7 | 9 | 9 | 7 | 0 | 0 | 6 | – | 0 |
| GFT-FEDO | 7 | 7 | 9 | 9 | 7 | 0 | 0 | 6 | 0 | – |

[a] Definitions of primary scenarios are found in Table 1.
[b] Identical results for all three Windows operating systems.

simulations with differences, the deviation in CWAM did not exceed 2 kg ha$^{-1}$. Considering that CWAM values often exceed 5000 kg ha$^{-1}$ and considering the likelihood of other causes for uncertainty in model simulations, this deviation due to compiler or operating system factors was of minor concern. For 10 of the 773 total simulations, deviations in CWAM were greater than 10 kg ha$^{-1}$, and the maximum deviation was 213 kg ha$^{-1}$. These larger deviations were confined to simulations for wheat, barley, bahia grass, and brachiaria. By examining the behavior of the model state variables that ultimately determine CWAM, we were able to identifying poor coding practices, explained below, that led to the larger CWAM deviations for these crops. Coding modifications and improvements that eliminate these discrepancies will ensure the robustness of the model and increase confidence for its use on a wider range of computer platforms.

As shown by our secondary simulation scenarios, numerical accuracy was not affected by antivirus status or RAM disk usage (Table 5). However, when the Intel compiler was used to compile the model without the floating point handling option ('-fpe0') on Windows, 14 simulations of CWAM and 5 simulations of ETCM were different from the corresponding simulations with '-fpe0' enabled. This highlights the importance of the '-fpe0' compiler option to appropriately handle floating point exceptions during model simulations. Results also demonstrated numerical discrepancies depending on the optimization flag used with the Intel, g95, and gfortran compilers. Usage of an '-O3' optimization flag provided identical results to the '-O2' flag with the g95 compiler and resulted in only three problematic CWAM simulations for the Intel compiler. More discrepancies were found with gfortran at the '-O3' optimization level. When '-O0', '-Od' or '-O1' optimization was used, there were from 29 to 67 CWAM simulations and from 1 to 8 ETCM simulations with numerical discrepancies for all three compilers as compared to '-O2' optimization. Not only are simulation results compiler dependent (Table 4), but they are also dependent on the compiler options used (Table 5). Thus, to maintain confidence in the numerical accuracy and repeatability of simulation results when a simulation model goes open-source, software development efforts must focus on minimizing the numerical discrepancies that result from alternative compilations of the model. At a minimum, model developers must provide the makefiles or communicate the compiler options they use for rigorous model evaluation against field data. This highlights a major difference between open-source simulation models and other open-source software projects, where the details of numerical processing are not as critical in light of the software's overall purpose.

**Table 5**
Number of simulations (out of 773 total) having different values for canopy weight at maturity (CWAM) and cumulative evapotranspiration at maturity (ETCM) for each secondary scenario as compared to its corresponding primary scenario.[a]

| Test | CWAM | ETCM |
|---|---|---|
| INT-XP32-AV | 0 | 0 |
| INT-XP64-AV | 0 | 0 |
| INT-XP32-FH | 14 | 5 |
| INT-XP64-FH | 14 | 5 |
| INT-WIN7-FH | 14 | 5 |
| INT-XP32-Od | 37 | 8 |
| INT-XP32-O1 | 37 | 7 |
| INT-XP32-O3 | 3 | 0 |
| G95-XP32-O0 | 62 | 8 |
| G95-XP32-O1 | 29 | 1 |
| G95-XP32-O3 | 0 | 0 |
| GFT-XP32-O0 | 67 | 6 |
| GFT-XP32-O1 | 37 | 3 |
| GFT-XP32-O3 | 21 | 4 |
| INT-XP32-RD | 0 | 0 |
| G95-XP32-RD | 0 | 0 |
| GFT-XP32-RD | 0 | 0 |

[a] Definitions of primary and secondary scenarios are found in Tables 1 and 2, respectively.

### 3.2. Computational speed

Table 6 provides the list of primary simulation scenarios sorted by mean computational speed (simulations s$^{-1}$). With average speeds of 8.3 simulations s$^{-1}$ and above, simulations were fastest when compiled with the Intel Fortran compiler and run on a Linux operating system. Relatively quick performance, 6.7 simulations s$^{-1}$ on average, was also obtained with the g95 compiler on Linux. The Intel Fortran compiler generally provided the fastest model for Windows operating systems with speeds around 6.0 simulations s$^{-1}$. In particular, the more recent Windows 7 operating system offered an advantage over the two Windows XP operating systems. Compiling with g95 on Windows offered a clear speed advantage over compiling with gfortran on Windows. It is interesting to note that simulations with the DSSAT 4.5 distribution version of the model on Windows were slower than the versions of the model that we compiled in-house using an identical Intel compiler. These differences may be caused by internal settings of the Intel compiler that optimize the code for the particular machine on which it is compiled. Simulations with the gfortran-compiled model under MinGW on all three Windows operating systems were substantially slower than other compiler and operating system combinations.

Minimizing the number of output files written to the hard drive increased computational speed by an average of 9.8 simulations s$^{-1}$ across all primary scenarios (Table 6). With full file output, simulations on the Linux operating systems were clearly faster than those on the Windows operating systems. With minimum file output,

**Table 6**
Computational speed (simulations per second) for the primary simulation scenarios.[a]

| | All simulations | | | CROPGRO only[b] | |
|---|---|---|---|---|---|
| | Mean (sim $s^{-1}$) | Min (sim $s^{-1}$) | Max (sim $s^{-1}$) | Full Output (sim $s^{-1}$) | Minimum Output (sim $s^{-1}$) |
| INT-UBUN | 9.0 | 2.5 | 34.8 | 12.9 | 21.6 |
| INT-FEDO | 8.3 | 2.3 | 31.8 | 11.8 | 21.1 |
| G95-FEDO | 6.8 | 2.1 | 22.5 | 10.9 | 15.9 |
| G95-UBUN | 6.7 | 2.1 | 23.5 | 10.8 | 15.8 |
| INT-WIN7 | 6.4 | 1.8 | 18.3 | 5.7 | 19.9 |
| INT-XP32 | 5.8 | 1.9 | 20.7 | 5.5 | 20.5 |
| GFT-UBUN | 5.7 | 1.5 | 23.0 | 8.5 | 16.0 |
| G95-WIN7 | 5.7 | 1.7 | 17.7 | 6.7 | 14.6 |
| D45-WIN7 | 5.6 | 1.7 | 16.6 | 5.4 | 17.2 |
| G95-XP32 | 5.6 | 1.7 | 19.1 | 6.9 | 15.0 |
| INT-XP64 | 5.2 | 1.4 | 19.1 | 4.3 | 20.3 |
| G95-XP64 | 5.0 | 1.7 | 17.9 | 4.7 | 15.0 |
| GFT-FEDO | 5.0 | 1.3 | 21.6 | 7.5 | 15.2 |
| D45-XP32 | 4.3 | 1.3 | 14.6 | 4.9 | 17.2 |
| D45-XP64 | 4.1 | 1.2 | 14.6 | 4.6 | 16.3 |
| GFT-WIN7 | 2.1 | 0.6 | 10.0 | 2.9 | 11.6 |
| GFT-XP32 | 2.0 | 0.5 | 11.7 | 3.0 | 11.9 |
| GFT-XP64 | 1.7 | 0.5 | 9.9 | 2.8 | 11.7 |

[a] Definitions of primary scenarios are found in Table 1.
[b] Simulations were conducted with the CSM-CROPGRO-Soybean model.

computational speed on Linux did not more than double. However, with minimum file output on the Windows operating systems, computational speed more than tripled in many cases. This suggests that the Linux operating systems managed file access more efficiently than the Windows operating systems. With minimum file output, simulations with the Intel Fortran compiler were over 5.0 simulations $s^{-1}$ faster than simulations with the g95 or gfortran compilers on the same operating system. This further supports the idea that the Intel Fortran compiler may internally optimize the code for the particular machine on which it is compiled. For the Intel models compiled without the '-fpe0' flag, the computational speeds were more comparable to that for g95 and gfortran (not shown). Thus, the '-fpe0' flag itself may be responsible for the optimized performance. When disk access was minimal, this was an advantage because CPU and RAM characteristics likely governed the speed of computation, and the code may have been optimized for those characteristics. However, when the model required full disk access, the less efficient disk handling of the Windows operating system overrode any advantage gained by optimizing to the specific machine. This highlights the need to understand the effects of compiler options on the computational speed of software programs. Finally, the frequency with which a software program accesses the file system is of greater concern on Windows than on Linux.

The change in computational speed for each secondary simulation scenario as compared to its corresponding primary scenario is given in Table 7. As expected, running the model with antivirus enabled reduced simulation speed as compared to simulations with antivirus disabled. An average speed reduction of 1.7 simulation $s^{-1}$ was observed. Removing the floating point handling flag ('-fpe0') with the Intel compiler resulted in slower code by about 1.0 simulations $s^{-1}$ as compared to the Intel model with '-fpe0' enabled. Likewise, the '-O2' optimization flag produced the fastest code with all three compilers, because use of other optimization flags tended to reduce computational speed. Fig. 1 shows the mean computational speed for model simulations on Windows XP using the four optimization settings for each of the three compilers. For Intel and g95, the '-O2' optimization flag provided the fastest simulations. For gfortran, the optimization flag did not have a large impact on speed, which may indicate limitations of the gfortran compiler when used under MinGW on Windows.

Since file output substantially reduced computational speed on the Windows systems (Table 6), we implemented a RAM disk as an alternative method to reduce hard drive access. For the Intel compiler on Windows XP, use of a RAM disk increased mean computational speed by 1.0 simulations $s^{-1}$ (Table 7), and computational speed for full output simulations was increased by 4.1 simulations $s^{-1}$ (not shown). These increases made the Intel simulations on Windows XP competitive with g95 simulations on Linux (Table 6). Again, the speed increase may be related to the '-fpe0' compiler flag optimizing the code specifically for RAM characteristics, such that simulations on a RAM disk would be expected to improve. Simulations with g95 or gfortran on a Windows RAM disk did not improve computational speed as compared to simulations from the hard drive. Implementation of RAM disks offer some computational advantages for the Intel compiler on Windows systems. However, it may be quite impractical to reinitialize the RAM disk and reload the model and associated input files each time the computer is restarted. Using the Intel compiler on Linux systems offers greater computational advantages than using Intel with RAM disks on Windows.

**Table 7**
Change in mean computational speed (simulations per second) for each secondary scenario as compared to its corresponding primary scenario.[a]

| Test | Δ Speed |
|---|---|
| INT-XP32-AV | −1.7 |
| INT-XP64AV | −1.7 |
| INT-XP32-FH | −1.2 |
| INT-XP64-FH | −0.8 |
| INT-WIN7-FH | −0.9 |
| INT-XP32-Od | −1.8 |
| INT-XP32-O1 | −0.3 |
| INT-XP32-O3 | −0.1 |
| G95-XP32-O0 | −0.7 |
| G95-XP32-O1 | −0.4 |
| G95-XP32-O3 | −0.4 |
| GFT-XP32-O0 | −0.1 |
| GFT-XP32-O1 | +0.1 |
| GFT-XP32-O3 | −0.1 |
| INT-XP32-RD | +1.0 |
| G95-XP32-RD | −0.7 |
| GFT-XP32-RD | −0.4 |

[a] Definitions of primary and secondary scenarios are found in Tables 1 and 2, respectively.

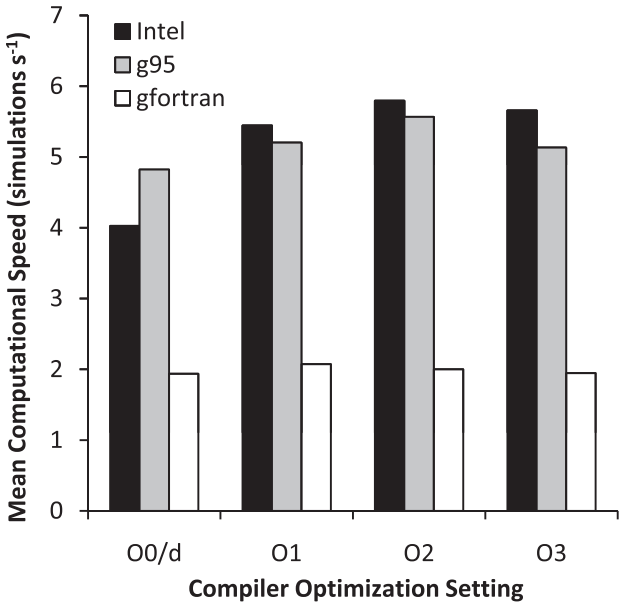


**Fig. 1.** Mean computational speed (simulations per second) of the DSSAT-CSM when compiled using four optimization settings with the Intel Fortran, g95, and gfortran compilers on Microsoft Windows XP.

Simulations of continuous cropping sequences for varying numbers of years illustrated additional issues affecting simulation speed (Fig. 2). Results are given for 35 iterations of the model compiled with Intel Fortran on Windows XP (INT-XP32). Simulations that specified minimal output to the hard drive were again substantially faster than those with full output. The standard deviation in computational speed among the 35 iterations for a given simulation length was not greater than 1.3 simulations $s^{-1}$, indicating a consistent performance no matter if full or minimum output was specified. These results further support the expectation that minimum output simulations are limited mainly by CPU and RAM performance characteristics. Slower computational speed for full output simulations indicate a dependence on slower computer system processes, particularly hard drive accessibility.

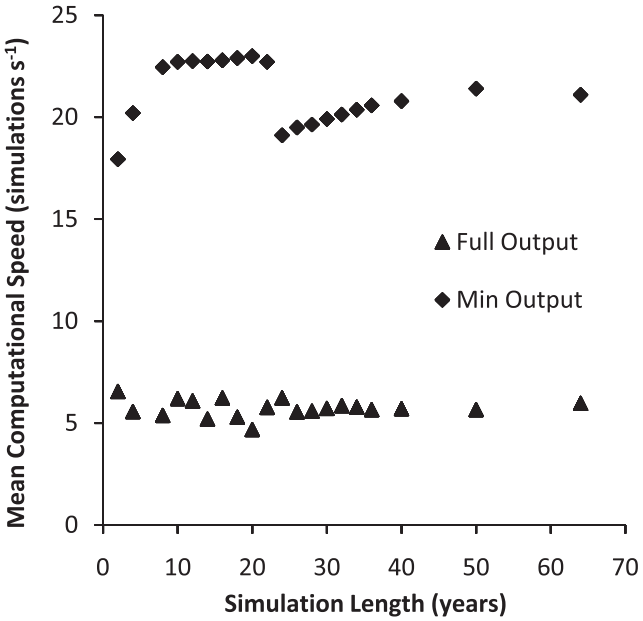


**Fig. 2.** Mean computational speed (annual simulations per second) versus the number of years of sequential simulations for 35 iterations with the DSSAT-CSM compiled with Intel Fortran on Microsoft Windows XP.

Minimizing simulation output resulted in computational speeds that followed a cumulative exponential distribution function as the simulation length increased from 2 to 22 years. Since the model reads multiple years of weather data at a time, there was an initial expense to accomplish this task. As the simulation length increased, there was a gradual increase in computational speed per simulation year because the weather data for the additional years had already been loaded in RAM. After 22 years, computational speed decreased, because the model had to fetch an additional block of weather data from the hard drive. Simulations with full output did not demonstrate a trend in computational speed with simulation length. Repeated hard drive access limited the speed of the full output simulations rather than any effect of the simulation length.

### 3.3. Lessons learned

The methods presented herein demonstrate a strategy for evaluating the performance of simulation models to prepare them for implementation using diverse compilers on multiple operating systems. The strategy has been used to improve the programming style, portability, robustness, and flexibility of the DSSAT-CSM. Several years ago, this study was initiated out of the lead author's desire to compile the DSSAT-CSM with open-source Fortran compilers and to utilize the DSSAT-CSM on Linux operating systems. Since DSSAT-CSM was not developed for use in these environments, neither gfortran nor g95 would compile the model, and both identified multiple errors throughout the code. Over a period of weeks, the lead author modified the code to address the issues deemed problematic by g95 and gfortran. Many of the issues involved the use of functions that were intrinsic to the Intel Fortran compiler, such as the end-of-file (EOF) function for testing whether a file pointer was at the end of a file. The EOF function was not intrinsic to g95 or gfortran and thus caused an error at compile time. As another example, the Intel compiler interpreted arguments for the 'GETARG' function, which is used to read command line arguments, differently than g95 and gfortran. The g95 compiler also halted compilation when the code specified the use of a REAL variable as the index for control structures ('DO' loops), while the Intel compiler permitted this practice. The simple solution was to change the data type for the control structure index variable to INTEGER. Another issue was the preference of the g95 compiler to have parentheses around negative exponents and negated variables. To address these and other issues, the code was modified in a way that would satisfy multiple compilers without affecting the overall functionality of the code. The result was a more portable and more versatile DSSAT-CSM.

In addition to detecting errors at compile time, software testing with multiple compilers is useful for detecting coding bugs at run time. For example, one of the simulations in the present study produced an access violation with the g95-compiled model but not with the Intel-compiled model. We were able to track the problem to a module that had not implemented the SAVE command. As a result, one of the array index variables was not properly saved, and this caused an exception when the model tried to access memory beyond the allocated bounds of the array. Another run-time error occurred when the g95-compiled model attempted to read a single decimal point ('.') from a model input file as a REAL variable. Such an operation was permissible with the Intel-compiled model, but not with the g95-compiled model. The solution was to modify the input file to contain actual real numbers, such as '-99' (for missing value) or '0.0' as appropriate. These examples further support the value of using multiple compilers to improve the portability, accuracy, and versatility of simulation models. In particular, we

found the g95 compiler to be substantially less forgiving than the Intel compiler. If the objective is to produce robust, versatile code, we deem this a positive quality.

Even when multiple compilers successfully compiled the code and the model was able to run without error, further areas for coding improvement were identified when comparing the simulation results for different compiled versions of the model (Table 4). As an example, we tracked the crop growth state variables for the simulation that resulted in the largest difference in CWAM (213 kg ha$^{-1}$) between the Intel- and g95-compiled versions of the model on the 32-bit Windows XP operating system. This issue occurred in the wheat growth module of the DSSAT-CSM. The root of the problem was identified in the leaf weight state variable (LFWT), which gives the weight of leaves on an individual wheat plant each day and was typically less than 2.0 g plant$^{-1}$ for this particular wheat growth simulation. Several coding issues were identified in conditional 'IF' statements where the LWFT variable was evaluated against quantities such as '0', '0.0', or '0.000001'. Due to differences in handling of floating point values between the compilers, some of these conditional statements were evaluated differently, thus resulting in different daily calculations of leaf weight. The problem was exacerbated by the fact that LFWT typically held a relatively small positive real number, such that differences in LFWT at the sixth to eighth decimal place had greater effect on the overall simulation. These differences were magnified by the nature of the model to add or subtract small amounts of mass to the LFWT variable repeatedly on a daily basis over the wheat growing season. Since the model uses the leaf weight of an individual wheat plant (LFWT) to scale up to CWAM (kg ha$^{-1}$) based on the planting density, it is easy to understand how a CWAM discrepancy of 213 kg ha$^{-1}$ is possible without careful floating point accounting in the LFWT state variable. One solution may be to discontinue the use of the REAL data type in the DSSAT-CSM code and instead define all floating point variables as DOUBLE PRECISION. This should eliminate differences in the way separate compilers handle the REAL data type. Use of multiple compilers can help to identify and correct poor coding practices that create model output variability that is solely compiler dependent. To maintain robustness of computer code used for scientific purposes, researchers and model developers must strive to minimize such problems, especially when model users have reason to implement the model with diverse compilers on multiple operating systems. The methodology presented herein can be used for this purpose.

Further suggestions for improvement of the DSSAT-CSM arose from its implementation on Linux operating systems. The main issue was the case sensitivity of the Linux file system, whereas the file system of Windows operating systems is not case sensitive. To run the model on the Linux systems, we had to insure that there were no case discrepancies between the file name strings generated in the code and the actual file name in the file system. The easiest way to handle this problem was to change the file name in the file system to match what the model expected. This is another important consideration to ensure model compatibility between Windows and Linux.

## 4. Conclusions

We have demonstrated a methodology for evaluating simulation models using various compilers on multiple operating systems. Using the DSSAT-CSM as an example case, we showed how the methodology can be used to broaden the model's applicability within multiple programming and computing environments, an important precursor to model development within the open-source paradigm. The methodology was also useful for improving model code to reduce numerical discrepancies between compilers and for identifying model implementation strategies that increase computational speed. Although the results focus specifically on the DSSAT-CSM, we believe the following methodology is applicable for improving the robustness and performance of other simulation models and software tools:

- Design a desktop computer system with a swappable hard drive rack to facilitate rapid testing of multiple operating systems.
- Use the Eclipse integrated development environment to identically manage diverse compilers on various operating systems.
- Assess model performance with multiple compilers and operating systems for your model and programming language of choice.
- Track model state variables that are regularly incremented by small floating point values to address issues of floating point handling and numerical accuracy between compilers.
- Improve the model code to reduce numerical differences resulting from alternate compilations.

This basic approach can be used to guide model development toward reliable implementation in multiple software environments.

For scientists who pursue computationally intensive modeling applications, our methodology can be used to facilitate model implementation in both Windows and Linux environments. This offers several computational advantages, such as faster performance on Linux desktop machines and ability to implement the model on Linux-based high performance computing clusters. For researchers who lack access to advanced computing systems, the following considerations will likely increase simulation speed on standard desktop computers:

- Minimize read and write operations to the hard drive, especially on Windows operating systems.
- Disable antivirus programs and other programs that interrupt disk access.
- Use strict floating point handling to halt programs on floating point exceptions.
- Implement Fortran programs with the Intel Fortran compiler on a Linux operating system.

The real value of testing simulation models with different compilers and operating systems is its ability to facilitate model implementation by a broader audience and for a wider range of computer environments. These assessments are readily extended to examine issues related to computational speed, which may identify bottlenecks that would not be anticipated by most users.

## Acknowledgements

The authors acknowledge USDA-ARS-ALARC technician, William Luckett, for building and maintaining the computer that was used to conduct this simulation study. We also thank Steve Welch, Lars Koesterke, and the anonymous reviewers for their constructive comments to improve this work. Finally, the authors acknowledge the countless researchers who have contributed to the development of the Fortran code for the DSSAT-CSM over the past several decades.

## References

Anothai, J., Patanothai, A., Jogloy, S., Pannangpetch, K., Boote, K.J., Hoogenboom, G., 2008. A sequential approach for determining the cultivar coefficients of peanut lines using end-of-season data of crop performance trials. Field Crops Research 108 (2), 169–178.

Boote, K.J., Jones, J.W., Hoogenboom, G.H., White, J.W., 2010. The role of crop systems simulation in agriculture and environment. International Journal of Agricultural and Environmental Information Systems 1 (1), 41–54.

Boote, K.J., Jones, J.W., Pickering, N.B., 1996. Potential uses and limitations of crop models. Agronomy Journal 88 (5), 704–716.

Braga, R.P., Jones, J.W., 2004. Using optimization to estimate soil inputs of crop models for use in site-specific management. Transactions of the ASAE 47 (5), 1821–1831.

Hauge, Ø., Ayala, C., Conradi, R., 2010. Adoption of open source software in software-intensive organizations – a systematic literature review. Information and Software Technology 52 (11), 1133–1154.

Hoogenboom, G., Jones, J.W., Wilkens, P.W., Porter, C.H., Boote, K.J., Singh, U., White, J.W., Hunt, L.A., Lizaso, J.O., Tsuji, G.Y., 2011. Development and application of the decision support system, DSSAT. In: Annual International Meeting Abstracts. American Society of Agronomy, Madison, Wisconsin.

Hoogenboom, G., Jones, J.W., Wilkins, P.W., Porter, C.H., Batchelor, W.D., Hunt, L.A., Boote, K.J., Singh, U., Uryasev, O., Bowen, W.T., Gijsman, A., Toit, A., White, J.W., Tsuji, G.Y., 2010. Decision Support System for Agrotechnology Transfer. Version 4.5. CD-ROM. University of Hawaii, Honolulu, HI.

Jones, J.W., Hoogenboom, G., Porter, C.H., Boote, K.J., Batchelor, W.D., Hunt, L.A., Wilkens, P.W., Singh, U., Gijsman, A.J., Ritchie, J.T., 2003. The DSSAT cropping system model. European Journal of Agronomy 18 (3-4), 235–265.

Keeling, K.B., Pavur, R.J., 2007. A comparative study of the reliability of nine statistical software packages. Computational Statistics and Data Analysis 51 (8), 3811–3831.

Knusel, L., 2005. On the accuracy of statistical distributions in Microsoft Excel 2003. Computational Statistics and Data Analysis 48 (3), 445–449.

Merali, Z., 2010. ERROR: why scientific programming does not compute. Nature 467 (7317), 775–777.

Mockus, A., Fielding, R., Herbsleb, J., 2002. Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology 11 (3), 309–346.

Nol, L., Heuvelink, G.B.M., Veldkamp, A., de Vries, W., Kros, J., 2010. Uncertainty propagation analysis of an $N_2O$ emission model at the plot and landscape scale. Geoderma 159 (1–2), 9–23.

Paz, J.O., Batchelor, W.D., Babcock, B.A., Colvin, T.S., Logsdon, S.D., Kaspar, T.C., Karlen, D.L., 1999. Model-based technique to determine variable rate nitrogen for corn. Agricultural Systems 61 (1), 69–75.

Quaife, T., Lewis, P., De Kauwe, M., Williams, M., Law, B.E., Disney, M., Bowyer, P., 2008. Assimilating canopy reflectance data into an ecosystem model with an Ensemble Kalman Filter. Remote Sensing of Environment 112 (4), 1347–1364.

Thorp, K.R., Batchelor, W.D., Paz, J.O., Kaleita, A.L., DeJonge, K.C., 2007. Using cross-validation to evaluate CERES-Maize yield simulations within a decision support system for precision agriculture. Transactions of the ASABE 50 (4), 1467–1479.

Thorp, K.R., Hunsaker, D.J., French, A.N., 2010. Assimilating leaf area index estimates from remote sensing into the simulations of a cropping systems model. Transactions of the ASABE 53 (1), 251–262.

Tsuji, G.Y., Hoogenboom, G., Thornton, P.K. (Eds.), 1998. Understanding Options for Agricultural Production. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Veith, T.L., Wolfe, M.L., Heatwole, C.D., 2003. Optimization procedure for cost effective BMP placement at a watershed scale. Journal of the American Water Resources Association 39 (6), 1331–1343.