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