Computers & Geosciences 75 (2015) 66–72

Contents lists available at ScienceDirect

Computers & Geosciences

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

OpenMP-accelerated SWAT simulation using Intel C and FORTRAN compilers: Development and benchmark

Seo Jin Ki [a], Tak Sugimura [b], Albert S. Kim [a,*]

[a] Department of Civil and Environmental Engineering, University of Hawaii at Manoa, Honolulu, HI 96822, USA
[b] College of Tropical Agriculture and Human Resources, University of Hawaii at Manoa, Honolulu, HI 96822, USA

ARTICLE INFO

Article history:
Received 12 April 2014
Received in revised form
28 October 2014
Accepted 30 October 2014
Available online 6 November 2014

Keywords:
Soil and Water Assessment Tool
iOMP-SWAT
OpenMP
Parallelization
Intel compilers
GNU utilities

ABSTRACT

We developed a practical method to accelerate execution of Soil and Water Assessment Tool (SWAT) using open (free) computational resources. The SWAT source code (rev 622) was recompiled using a non-commercial Intel FORTRAN compiler in Ubuntu 12.04 LTS Linux platform, and newly named iOMP-SWAT in this study. GNU utilities of `make`, `gprof`, and `diff` were used to develop the iOMP-SWAT package, profile memory usage, and check identicalness of parallel and serial simulations. Among 302 SWAT subroutines, the slowest routines were identified using GNU `gprof`, and later modified using Open Multiple Processing (OpenMP) library in an 8-core shared memory system. In addition, a C wrapping function was used to rapidly set large arrays to zero by cross compiling with the original SWAT FORTRAN package. A universal speedup ratio of 2.3 was achieved using input data sets of a large number of hydrological response units. As we specifically focus on acceleration of a single SWAT run, the use of iOMP-SWAT for parameter calibrations will significantly improve the performance of SWAT optimization.

1. Introduction

Sustainable watershed management enhances security of national water resources, and promotes efficient water use, human health, and ecosystem services (US EPA, 2008). The Soil and Water Assessment Tool (SWAT), released by the Agricultural Research Service in the United States Department of Agriculture (Arnold et al., 1993), is a popular model to predict water flow and pollutant transport at the watershed level (Gassman et al., 2007). After its initial release, SWAT has been widely applied to multi-scale hydrological modeling from field-scale scenario analysis (e.g., septic systems and Best Management Practices) to regional/continental-scale impact assessment of climate change (Douglas-Mankin et al., 2010; Tuppad et al., 2011). SWAT addresses various water security issues (such as peak flow, soil erosion, and nutrient load) for alternative land management practices (Krysanova and Arnold, 2008). Results of SWAT simulations can be used to develop effective watershed management strategies under the Total Maximum Daily Loads program (Gassman et al., 2007; US EPA, 2008). In addition, SWAT can be easily integrated with other software such as SWMM and MODFLOW for the sake of holistic environmental modeling (Tuppad et al., 2011). Extended SWAT models, e.g., SWAT-G and SWAT-BF, now improve estimation capabilities of local watershed processes (Gassman et al., 2007; Tuppad et al., 2011).

While the SWAT model is widely applied to watershed management, its slow performance (due to intensive input/output operations, frequent subroutine calls in a nested hierarchy, retrieval of irrelevant details from each sub-basin) hampers large-scale simulations and fast parameter calibrations. In particular, the following gives more details on slow model performance. First, all input/output files are stored at ASCII text (not binary) format in a single directory under TextInOut. These are kept as individual files, which are not integrated with respect to file types or variable characteristics. The number of total files is, therefore, linearly proportional to the number of hydrological units as well as time durations. Second, SWAT does not have sophisticated data processing features of periodic and real-time inputs. Processing time of daily input data is continuously elongated as the total simulation time keeps increasing. Third, SWAT is not well established in structured and object-oriented programming paradigms. So, the whole memory space is accessible to any subroutine regardless of the hierarchical levels during a simulation.

In recent environmental modeling research, parallel computing has become important due to the availability of high performance computing resources, which allow researchers to run large-scale simulations (Wood et al., 2011). Hydrological modeling studies have a wide span from (inter-dependent) surface and subsurface hydrology (Cheng et al., 2005; Li et al., 2011; Wu et al., 2002,Vivoni et al., 2011; Yeh et al., 2011) to coastal processes (Wai and Lu,

* Corresponding author. Fax: +1 808 956 5014.
E-mail address: albertsk@hawaii.edu (A.S. Kim).

http://dx.doi.org/10.1016/j.cageo.2014.10.017
0098-3004/

2000). Computational hydrology aims to accelerate overall execution time of complex models using parallel computation (Laloy and Vrugt, 2012; Vrugt et al., 2006; Zhang et al., 2013). Recent developments include the use of the script languages Python and R, bounded with message passing interface (MPI) libraries (Joseph and Guillaume, 2013; Wu et al., 2013; Zhang et al., 2013).

Apostolopoulos and Georgakakos (1997) introduced a parallelization concept for predicting streamflow of high order and long river reaches. Recently, Li et al. (2011) achieved the maximum 5.34 times speedup using an effective sub-basin partitioning method, and Vivoni et al. (2011) obtained more than 70 times acceleration by efficiently maintaining a load balance and data communication. Parallelized SWAT using 4–8 processors noticeably reduced the execution time, but a linear speedup was hardly achieved when a large number of processors were employed (Rouholahnejad et al., 2012; Zhang et al., 2012). This is because the computational speedup is strongly correlated with not only parallel algorithms and inter-process communications (Tang et al., 2007; Wai and Lu, 2000), but also hardware configurations of shared- or distributed-memory architectures (Rouholahnejad et al., 2012; Zhang et al., 2012). Wai and Lu (2000) indicated that numerical schemes and solvers affected the efficiency and flexibility of parallel modeling, and Zhang et al. (2012, 2013) reported that parallel performances also depended on input data structures. Hardware architectures tailored to specific models must have equal importance in achieving the ideal speedup, which in turn enhances the parallel scalability.

Several parallel algorithms were used to accelerate the slow calibration process of SWAT, which normally took a few hours to several days. Rouholahnejad et al. (2012) implemented parallel algorithms in Sequential Uncertainty Fitting (ver 2) for optimized calibration and SWAT-CUP for uncertainty analysis, and obtained the almost ideal speedup using 8 processors. Zhang et al. (2013) parallelized a multi-objective optimization method (i.e., AMALGAM) using a Python with MPI and OpenMPI, and obtained 45–109 times speedup using a computer cluster consisting of 2512 processors and 13.6 TB memory, which depended on task complexity, i.e., the size and time scale of simulation data sets. However, the actual speedup benefits from this package appeared to be significantly less than estimated because standard computers with multiple processors completed distributed serial runs faster than a single core. In other words, execution time of a reference serial run (in the computer cluster) itself was overestimated. Zhang et al. (2012) evaluated the efficiency of parallel performance of an evolutionary multi-objective optimization method using a 16-core machine with 24 GB of memory, and reported that the speedup was almost linear using 8 cores or less. Wu et al. (2013) employed a load balancing of the standard master-slave scheme for the SWAT parallelization, and obtained the maximum speedup of 3.36 times when 5 cores (among 8 logical cores) were used in a workstation with 4 GB of memory. Recently, uncertainty analysis of the SWAT model was conducted using the R language that parallelized the likelihood computations (Joseph and Guillaume, 2013).

Important issues for parallel SWAT development and execution are as follows. First, there is the critical number of cores, above which the speedup becomes a pseudo-constant. The speedup ratio seems to be bounded to the order of O(1) and it hardly exceeds 10 times with 8–16 cores. Note that although Zhang et al. (2013)'s work exceeded 10 times speedup, a low net speedup per core was achieved when the extraordinarily large number of cores (2512) were used. Second, parallel results have not been thoroughly compared with that of a serial run using the same parameter settings. This is because the calibration aims to provide optimal parameters to minimize modeling errors. When parallel algorithms are used, total time required to complete a calibration run depends on several factors such as load balance, search algorithms, initial values, and tolerance errors. Due to the arbitrariness of stop conditions of SWAT calibration, it is hard to accurately estimate speedup of parallel SWAT against a serial run. So, the optimal parameter settings (of the Pareto front) recommended from a parallel calibration may not be identical to those of a serial calibration. Several sets of parameter values can give similar modeling results. Third, the speedup strategy of a single SWAT run has never fully studied, to the best of our knowledge. SWAT subroutines need to be investigated in terms of their individual runtimes during the single run. If the serial run is accelerated a few times, then the parallel calibration must be much faster and more manageable.

We aim to, therefore, provide an achievable speedup method for the fast SWAT execution using open (free) computational resources in affordable hardware systems. In this study, we (i) analyzed the software structure of the latest SWAT code (rev 622), (ii) identified primary bottlenecks in the program execution, and (iii) developed an acceleration method for SWAT executions. Open resources include the Linux OS (Ubuntu 12.04 LTS), GNU `make`, `gprof`, and `diff` utilities, and non-commercial Intel C and FORTRAN compilers with Open Multiple Processing (OpenMP) library. In addition, we discuss the future parallelizability of the SWAT code in terms of its software hierarchy and subroutine calls, hoping the current work provides fundamental insight into future software engineering of the SWAT program. The source code sets of iOMP-SWAT and `Makefile` will be available on the corresponding author's web site: www.albertsk.org.

2. Simulation

2.1. Computational resources: software and hardware

We used the Ubuntu (12.04 LTS) Linux as a computing platform, which recently gained popularity ahead of Debian.[1] All GNU packages and other community-developed programs are abundantly available and fairly manageable by using the **apt-get** command. GNU supports C/C++ (`gcc/g++`) and FORTRAN (`gfortran`) compilers with conservativeness and compatibility. When Intel chip-based hardware is available for computational research, commercial Intel compilers are often preferred because of their superior performance, as pre-optimized with CPUs. Intel Corporation had non-commercial releases of C/C++ and FORTRAN compilers only for a Linux platform, which were used in the current research.

Three standard GNU utilities were used for our software development: `make`, `gprof`, and `diff`. SWAT (rev 622) uses a mixed grammar style of FORTRAN 77 and 90. To systematically compile SWAT source codes with proper options, we used `make` utility, preparing a `Makefile` to instruct compiling sequences and subroutine dependencies. GNU `gprof` referred to a profiling utility to analyze software, which was used in this study to find the slowest subroutines during the SWAT runs. We used the `diff` utility to compare output files of the original SWAT and iOMP-SWAT runs. If two files (of the same extension of the SWAT) are identical to each other, `diff` utility shows an empty message in the standard output. Two development criteria of iOMP-SWAT were (1) universal acceleration over the original SWAT and (2) identicalness of simulation results to the original SWAT (rev 622).

Our affordable hardware system consisted of two quad-core Intel® Xeon® E5345 2.33 GHz CPUs with 16 GB of total memory. Processor architecture is X86_64 and CPUs are operated in 64-bit

[1] ⟨http://w3techs.com/blog/entry/debian_ubuntu_extend_the_dominance_in_the_linux_web_server_market_at_the_expense_of_red_hat_centos⟩.

mode. The system has two sockets, and each has four cores. The eight cores have identification numbers 0–7, and each has internal 32 kB L1 cache memory. L2 cache memory (4096 kB) is shared between paired cores, such as (0,1) and (2,3) in socket 0, and (4,5) and (6,7) in socket 1. Two 8 GB Hynix FBD DDR2-667 synchronous memories are connected via an Intel 5000P memory controller.

2.2. Composition and compilation of SWAT subroutines

Three key files of SWAT are as follows: (1) a main program file (*main.f*) to generate an executable file (main.x), (2) a module file (*modparm.f*) containing a **parm** module declaring variables and arrays, and (3) a dynamic allocation routine (*allocate_parms.f*). The main file includes *modparm.f*, and governs sequential procedures of the SWAT simulation. The executable file reads input files, calls subroutines, and finally writes output files. In manually building the SWAT package, *modparm.f* should be pre-compiled to generate **parm.mod** (a binary module file of all the pre-declared variables). The SWAT code set consists of 299 files of FORTRAN77 (of extension.f) and three newly added FORTRAN90 (.f90) files (NCsed_leach.f90, carbon_zhang2.f90, and hruday.f90). With few exceptions, a file name is identical to a subroutine contained inside.

2.3. Hierarchy of SWAT subroutines

The main program calls 27 subroutines, and **simulate** is one of them, which calls the **command** subroutine: **command** is not included in *main.f* file, but is in the **simulate** subroutine file. We let the subroutines be included in *main.f* be at level 1, and therefore **simulate** and **command** lie at level 2 and 3, respectively. Two subroutines **nuts** (in *nuts.f*) and **nfix** (in *nfix.f*) are at the lowest level 7. Memory of the SWAT software is globally open to the public. All the variables (and arrays) declared in the **parm** module are accessible from subroutines at any levels (1 to 7). A total of 286 subroutines have a header part containing a line of

```
use parm
```

which brings all the declared variables to local subroutines.

FORTRAN 77 has COMMON blocks to share information across subroutines, while F90 uses modules, which may provide extra convenience in programming. In general, information can be shared or passed between subroutines. The former is the case of the SWAT, and the later requires a list of specific arguments. Each SWAT routine file includes the routine purpose and a list of incoming and outgoing variables. To increase data privacy and to share only necessary information, the "use parm" can be changed to

```
use parm, only: var1, var2, …
```

where `var1` and `var2` are arbitrary variable names. If OpenMP is used, (paired) cores will compete with each other to occupy the necessary memory at L2 cache level, which may take a comparable amount of time for arithmetic operations.

2.4. Classification of sequential SWAT simulation

The main part of the SWAT execution starts by calling the **simulate** subroutine which consists of three nested DO loops: yearly (do curyr=1, nbyr) and daily (do i=id1, idlst), under which HRU loops (do j=1, mhru or do j=1, nhru) exist. For testing purposes, we reversed sequences of yearly and daily DO loops (one at a time) such as

```
do curyr=nbyr, 1, −1
```

and

```
do i=idlst, id1, −1
```

GNU `diff` utility gave an enormous amount of messages, indicating that the two output sets were heavily heterogeneous. This is because the result from the last time step is an input to the current time step. Interestingly, reversed HRU loops such as

```
do j=mhru, 1, −1
```

gave the same result as the forward and reversed serial runs. This result demonstrates that parallelization of the single SWAT run should be implemented under the daily time loop, where the software hierarchy and subroutine dependencies become sophisticated.

2.5. Sample data sets

To develop the fast iOMP-SWAT code for various hydrologic scales, we prepared Maui and Upper Colorado data sets at the sub-watershed level and at the regional level, respectively, in terms of a hierarchical hydrologic unit system (see Fig. 1). The hydrologic unit refers to a unique identifier consisting of numerical digits, which classify the national hydrologic features in the US (e.g., drainage basins including rivers and streams) into 6 different geographic levels, from region (2-digit code) to sub-watershed (12-digit code).

Depending on geographic scales, digital elevation models (DEMs) at 30 m and 100 m resolutions were used to digitize stream networks for Kalialinui Gulch Sub-watershed and Upper Colorado Region, resulting in 29 and 143 sub-basins with threshold areas of 100 and 139193 ha, respectively. The same resolutions were applied to the land use and soil definitions (i.e., detailed vs general soil maps) with slope classification from DEMs. Total 110, 447, and 989 HRUs were produced for Kalialinui Gulch Sub-watershed, and 3843, 5689, 10993, and 26705 HRUs were created for the Upper Colorado Region by changing percentage thresholds in land use, soil, and slope layers. The simulation period was set to 10 and 30 years for Kalialinui Gulch Sub-watershed and Upper Colorado Region, respectively. Default weather data available in the SWAT program (across the US) were provided during the simulations.

2.6. Methods

2.6.1. Reference output from serial run

In our work, we first compiled the original SWAT program using an Intel FORTRAN compiler, ran the standard simulation in the serial mode, and stored output files in a separate directory. All serial output files were designated as reference results to be compared with those of iOMP-SWAT. Among various input data sets, we selected the case study of Upper Colorado having 26705 HRUs as our first reference serial run. The full data set was prepared for 30 years, for the period between 1980 and 2009, and we used only the first 3-year data set for iOMP-SWAT to finish test runs within a reasonable amount of time (10–20 min). We used a Linux command **time** to measure the time spent for iOMP-SWAT execution, and "user time" measured by this command was recorded as overall runtime. We added two FORTRAN calls, CPU_TIME (T1) and CPU_TIME (T2), in the main program to measure file-reading time, the difference between T2 and T1. The overall runtime and file-reading time for the single serial run were measured as 756.33 and 9.63 s, respectively, indicating that the serial execution of the original SWAT only spent 746.70 (=756.33−9.63) s for arithmetic operations.

2.6.2. Identification of the slowest subroutines using profiling (`gprof`)

We used compiling option "-pg" for memory profiling, and named the executable file **iomp_swat662**. When the execution was done, a profile 'gmon.out' was generated and information was retrieved using the `gprof` command. Fig. 2a presents important portions of the profile, indicating that **intel_memset**, **irr_rch**, **nminrl**, and **pminrl2**

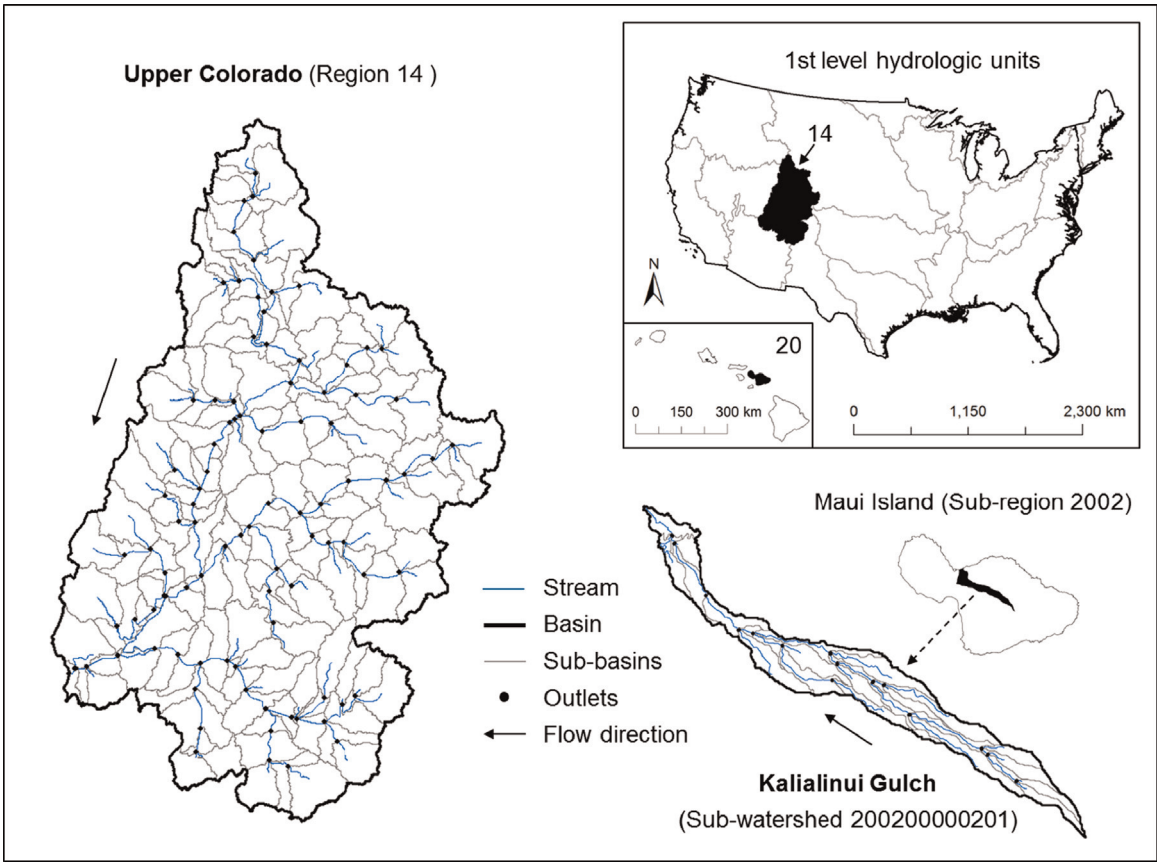

Fig. 1. Input data sets for SWAT runs: (large-scale) Upper Colorado Region and (small scale) Kalialinui Gulch on Maui island, Hawaii. The former is ranked at the top-level hierarchy of hydrologic units (see 2-digit code 14), whereas the later is compiled from the lowest level of hydrologic units (see 12-digit code 200200000201).

are very time-consuming routines. The least-time consuming routines were **writeaa** and **subyr**, and the main program itself barely took any time. The cumulative time (755.25 s) was close to the "user time" measured by the **time** command. On the other hand, routine of **intel_memset** was not found in the SWAT package.

Accelerated execution by parallelizing **irr_rch**, **nminrl**, and **pminrl2** routines does not seem to be promising. First, the total time spent by all three subroutines was 85.08 (=39.94+23.80+21.34) s, which was only 11% of the overall runtime. Second, these subroutines include a number of individual and/or multiple, nested DO loops of three- or four-dimensional arrays. This heavily index-oriented calculation (i.e., the starting index in the lowest level subroutine first and the ending last, then the next high-level subroutine and so on) is hard to parallelize because hierarchical relationships are often not independent but sophisticatedly linked across the subroutine hierarchy of the nested loops. Due to this reason, unit jobs cannot be flexibly allocated on various nodes. Third, variables are not passed as arguments, but fully shared at any subroutine level. In our opinion, the original SWAT is hard to parallelize because of the multiple hierarchical levels, subroutine granularity, and excessively open memory structures. As shown above, the most time-consuming routine was reported as **intel_memset**, a non-SWAT routine. We found that it originated from the **memset** function of C language embedded in Intel compilers. The purpose of this function is to rapidly set variables to zero. Surprisingly, the original SWAT program spent as much as 56% (423 out of 756 s) of the overall runtime to nullify variables and arrays (see Fig. 2a). In C language, **memset** is an intrinsic function to be called. In FORTAN, this is not explicit, but deeply embedded at the compiler level. For instance, one can use "myarray=0." in f90, where "myarray" is an arbitrary array. To call **memset** function in SWAT FORTRAN, we made a C wrapping function and named it fmemset.c, as follows:

```
#include <string.h>
#file name: fmemset.c
void fmemset_(void *a, int *n) {
memset(a,0,*n);}
```

where the underscore after fmemset is to make this C function be called from a FORTRAN routine. We tested this routine with a number of 2D and 1D arrays, and observed that execution speed was improved about 10–50%. The fmemset performance depended on array dimensions and sizes, but overall it was faster than the standard way.

We found that **allocate_parms** called **zero0**, **zero1**, **zero2**, **zero_urbn**, and **zeroini** only one time when the simulation started, and these zero-setting routines spent only 0.1 seconds or less. On the other hand, **varinit** was called by **subbasin**, and **command** called **subbasin**. The hierarchical sequence of calling **varinit** was: **main** (first level), **simulate** (second level), **command** (third level), **subbasin** (fouth level), and **varinit** (fifth level).

As discussed above, the **command** routine is called in the daily loop of the **simulate** routine. Within the hydrographic loop of **command** routine, **subbasin** is called depending on routing cases (if routing command code, icodes=0), and finally **varinit** is called within a loop of **subbasin**. Therefore, the **varinit** routine must be called a number of times per day during yearly SWAT simulations. **varinit** subroutine contains more than 100 variables to be set to zero and among them **hhsedy** is the only 2D array while others are scalars and 1D arrays. We changed the nullifying statement of "hhsedy=0." by calling fmemset for "hhsedy". In addition, we assigned nullifying tasks to four (real) threads of OpenMP using FORTRAN `select case` statement. We used only the unpaired cores of ID=0, 2, and 4 among 8 physical cores. SWAT variables

a

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 55.99    422.89   422.89                             __intel_memset
  5.29    462.83    39.94                             irr_rch_
  3.15    486.63    23.80                             nminrl_
  2.83    507.97    21.34                             pminrl2_
...
  1.07    614.41     8.05                             varinit_
...
  0.00    755.25     0.01                             writeaa_
  0.00    755.25     0.01                             subyr_
  0.00    755.25     0.00        1     0.00     0.00  MAIN__
```

b

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 12.14    39.76    39.76                              irr_rch_
  7.21    63.38    23.62                              nminrl_
  6.86    85.85    22.47                              pminrl2_
  6.06   105.70    19.86                              sumv_
...
  2.23   196.58     7.31                             varinit_
  2.17   203.69     7.11                             surfstor_
  2.05   210.42     6.73                             __intel_memset
...
  0.00   327.62     0.00        1     0.00     0.00  MAIN__
```

Fig. 2. GNU gprof profiling analysis of the Upper Colorado data set (at a large geographic scale) for the 3-year simulation period of (a) the original SWAT and (b) iOMP-SWAT runs.

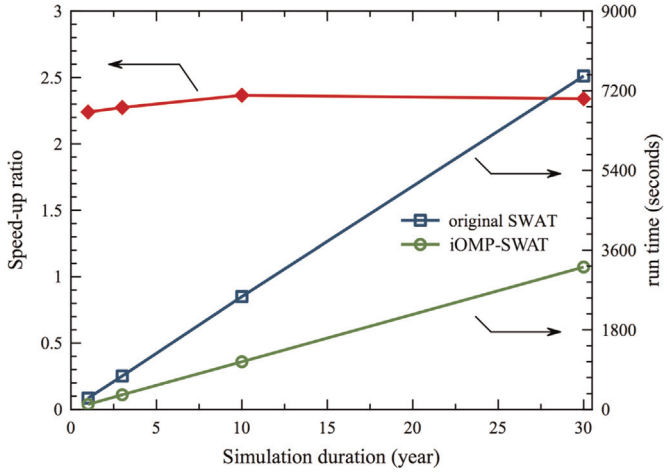

Fig. 3. Performance of iOMP-SWAT and the original SWAT using the Upper Colorado data set. Squares and circles indicate the overall runtime of the original SWAT and iOMP-SWAT executions, respectively (see the right axis), and the speedup (defined as their ratio) is shown using diamond symbols (see the left axis). (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

were divided into three groups with respect to their dimensions, and 0D (scalars), 1D, and 2D arrays were assigned to the core 0, 2, and 4. Note that cores (0,1) and (2,3) are paired in socket 0, and (4,5) and (6,7) are paired in socket 1. The unpaired cores (i.e., 0, 2, and 4) did not compete with each other to occupy memory spaces at local cache levels.

When all eight physical threads were used for zero setting in **varinit**, we experienced slower performance of the OpenMP SWAT version than the reference serial run. After we modified **varinit** using the three unpaired cores (ID=0, 2, and 4), we obtained very different profiling data from Fig. 2a, as shown in Fig. 2b. While runtime of **varinit** only minimally decreased from 8.05 to 7.31 s, that of **intel_memset** time significantly reduced from 422.89 to 6.73 s. If we considered the zero setting speedup only, it was 62.84 (=422.89/6.73) times. This is a very unexpected outcome using Intel C/FORTRAN cross compiling. If this zero-setting process is forced to use only one core of CPU, it would take less than 20.19 (=6.73×3) s, because 6.73 s is the runtime of the slowest core among the three. Virtual run of our OpenMP codes on a single CPU will take no longer than 20.19 (if identical hardware is used) and the speedup must be at least 20.95 (=62.84/3) times. More importantly, the overall iOMP-SWAT runtime was reduced from 756 to 327 s: the speedup was more than twice, i.e., 2.3 times. Note that we only modified **varinit** routine to use fmemset function (only for hhsedy) in the OpenMP mode. Among the eight available cores, only three of them actually participated in the zero setting process. Because the cores should not be paired, at least 6 cores are necessary (as a minimum) for this iOMP-SWAT execution. While cores 0, 2, and 4 were working, their partner cores 1, 3, and 5 were idle.

3. Results and discussion

3.1. Case study I: Colorado

We tested the iOMP-SWAT using two case groups: Upper Colorado and Maui, Hawaii. We used the same Upper Colorado data set with various durations of simulation years. For the initial

Table 1
A summary of detailed execution time for the Upper Colorado data set during various simulation periods using the original SWAT and iOMP-SWAT.

Years	Original SWAT (s)			iOMP-SWAT (s)			Speed-up ratio[a] (dimensionless)	
	Overall runtime	Pre-processing	Hydrologic simulation	Overall runtime	Pre-processing	Hydrologic simulation	Overall runtime	Hydrologic simulation
1	256.67	9.52	247.15	114.56	11.34	103.22	2.24	2.39
3	756.33	9.63	746.70	332.62	11.49	321.13	2.27	2.33
10	2551.95	9.90	2542.05	1078.62	11.38	1067.24	2.37	2.38
30	7534.62	9.69	7524.93	3219.38	11.50	3207.88	2.34	2.35

[a] The ratio of the overall runtime in the original SWAT to that iOMP-SWAT is estimated as the speed-up ratio for the overall runtime (see diamond symbols at red color in Fig. 3). The same is applied to the speedup ratio for the hydrologic simulation.

a

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  8.55      3.24     3.24                             nminrl_
  6.40      5.66     2.42                             pminrl2_
  5.89      7.89     2.23                             sumv_
  5.84     10.10     2.21                             __intel_memset
...
  2.56     21.65     0.97                             irr_rch
...
  0.01     37.84     0.01                             noqual_
  0.00     37.84     0.00        1     0.00     0.00  MAIN__
```

b

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 10.05      3.57     3.57                             nminrl_
  6.28      5.80     2.23                             pminrl2_
  6.11      7.97     2.17                             expf.L
  5.92     10.07     2.10                             powf.L
...
  0.51     30.32     0.18                             __intel_memset
...
  0.03     35.49     0.01                             xmon_
  0.00     35.49     0.00        1     0.00     0.00  MAIN__
```

Fig. 4. GNU gprof profiling analysis of the Maui data set (at a small geographic scale) for the 10-year simulation period of (a) the original SWAT and (b) iOMP-SWAT runs.

method development, we used 3 years of data from 1980. In addition to that, we tested iOMP-SWAT performance of the same data set for 1, 10, and 30 years.

The overall runtimes linearly increased with respect to the number of years in both the original SWAT and iOMP-SWAT executions (see Fig. 3). The iOMP-SWAT speedup ratio is defined as the runtime ratio of the original SWAT to iOMP-SWAT, as shown in Fig. 3 (see red line). For the simulation periods of 1, 3, and 10 years, the speedup ratio slightly increased to 2.24, 2.27, and 2.37, respectively, and that of year 30 was 2.34. The average speedup ratio was 2.31 ± 0.06 (standard deviation, SD). In each run, we separately measured the file-reading time by calling CPU_TIME () routine twice in the main program. The original SWAT and iOMP-SWAT took 9.69 ± 0.16 (SD) and 11.43 ± 0.08 (SD) s on the average, respectively (see Table 1). Although iOMP-SWAT spent more time to read all the input files, the standard deviation was less than that of the original SWAT. We do not have a clear explanation for the reason why iOMP-SWAT takes a longer time to read the same number of input files. Considering the overall runtime, the file reading time was negligible for yearly simulations in both SWAT runs. For the 1 year simulation of the original SWAT, file-reading time was less than 4% of the overall runtime. For multiple year simulations, this percentage will be smaller. The overall speedup ratios are similar with and without including the file reading times.

3.2. Case study II: Maui, Hawaii

Next, we tested iOMP-SWAT on small Hawaiian stream, which was located in Kalialinui Gulch Sub-watershed, Maui Island, Hawaii. The data set for this second test group was prepared for 10 years (from 2000 to 2009). The original SWAT run took only 37.84 s, so we simply compared performance of iOMP-SWAT and the original SWAT for this 10-year period only. Because the region

had only 989 HRUs at maximum, the SWAT program did not require memory space as large as the Upper Colorado case. The zero setting did not take too much time. As expected, **intel_memset** was ranked fourth, spending only 2.21 s, i.e., only about 6% of the overall runtime (see Fig. 4a). Interestingly, **irr_rch** routine took 0.97 s (ranked 12th), whereas in the Upper Colorado case, it was ranked first, followed by **nminrl** and **pminrl2**. The changes in relative subroutine ranks between the two data sets clearly imply that the SWAT performance strongly depends on hydrologic characteristics stored in the data sets, which conditionally determine sequences and frequencies of subroutine calls.

We applied iOMP-SWAT to the same Maui data set, and the overall runtime was 35.49 s (see Fig. 4b). The elapsed time for **intel_memset** was further reduced from 2.21 to 0.18, saving 2.03 s when iOMP-SWAT was used. This is almost equal to the overall runtime difference (i.e., 2.35 s) between 37.84 (of the original SWAT) and 35.49 s (of iOMP-SWAT). The speedup ratio of **intel_memset** only was calculated as 2.21/0.18 = 12.27, which was quite high, but still much less than that of the Upper Colorado case (62.84 for the 3-year simulation). The simulation results with a small data set (at sub-watershed level in Hawaii) for 10 years indicated the zero setting process was not a major task in the SWAT execution. Different strategies other than the zero setting must be needed to accelerate small-scale SWAT runs. In general, 20% of the overall runtime was spent for physical simulations such as irrigation (**irr_rch**, when the water source is a reach), mineralization/immobilization (**nminrl**), and phosphorous flux (**pminrl2**) for the Maui case. Therefore, these three routines also need to be actively investigated and revised for future SWAT releases, specifically for medium- to large-scale data sets.

4. Concluding remarks

We developed an iOMP-SWAT package from the original SWAT of rev 622. Open computational resources were used for this study, which included Ubuntu 12.04 LTS OS installed on an Intel-based hardware, non-commercial Intel C and FORTRAN compilers, and GNU utilities. Using GNU profiling utility `gprof`, we found that a significant portion of SWAT runtime was solely devoted to the zero setting of large arrays. In this light, calling fmemset function in the OpenMP mode noticeably reduced the overall runtime (on the average 2.3 times) of the large-scale Colorado data set, but only slightly for the small Maui data set in terms of HRUs.

In every test run, we checked identicalness of simulation results between iOMP-SWAT and the original SWAT using GNU `diff` utility. As the two results were perfectly identical to each other, `diff` command did not show any warning messages. This equivalence test, in our opinion, needs to be the primary requirement to have equal scientific rigor and robustness of future parallel versions of the SWAT program. Since SWAT is an event-based program, the next improvement using object-oriented programming is highly desired.

Acknowledgment

This research was funded by the Agricultural Research Service in the United States Department of Agriculture through the research project entitled “Parallelization of SWAT Model” (cooperative agreement no. 59-6206-3-004). The hardware was supported by the corresponding author's previous grant: the Faculty Early Career (CAREER) Development Program in the United States National Science Foundation (CBET04-49431). The corresponding author would like to specially thank Prof. Henri Casanova at the University of Hawaii at Manoa, for illuminating discussion of the fast zero setting and providing the C wrapping function.

References

Apostolopoulos, T.K., Georgakakos, K.P., 1997. Parallel computation for streamflow prediction with distributed hydrologic models. J. Hydrol. 197 (1–4), 1–24.
Arnold, J.G., Allen, P.M., Bernhardt, G., 1993. A comprehensive surface-groundwater flow model. J. Hydrol. 142 (1–4), 47–69.
Cheng, C.T., Wu, X.Y., Chau, K.W., 2005. Multiple criteria rainfall-runoff model calibration using a parallel genetic algorithm in a cluster of computers. Hydrol. Sci. J.—J. Des. Sci. Hydrol. 50 (6), 1069–1087.
Douglas-Mankin, K.R., Srinivasan, R., Arnold, J.G., 2010. Soil and water assessment tool (SWAT) model: current developments and applications. Trans. ASABE 53 (5), 1423–1431.
Gassman, P.W., Reyes, M.R., Green, C.H., Arnold, J.G., 2007. The soil and water assessment tool: historical development, applications, and future research directions. Trans. ASABE 50 (4), 1211–1250.
Joseph, J.F., Guillaume, J.H.A., 2013. Using a parallelized MCMC algorithm in R to identify appropriate likelihood functions for SWAT. Environ. Modell. Softw 46, 292–298.
Krysanova, V., Arnold, J.G., 2008. Advances in ecohydrological modelling with SWAT—a review. Hydrol. Sci. J.—J. Sci. Hydrol. 53 (5), 939–947.
Laloy, E., Vrugt, J.A., 2012. High-dimensional posterior exploration of hydrologic models using multiple-try DREAM(ZS) and high-performance computing. Water Resour. Res. 48 (1), W01526.
Li, T.J., Wang, G.Q., Chen, J., Wang, H., 2011. Dynamic parallelization of hydrological model simulations. Environ. Modell. Softw. 26 (12), 1736–1746.
Rouholahnejad, E., Abbaspour, K.C., Vejdani, M., Srinivasan, R., Schulin, R., Lehmann, A., 2012. A parallelization framework for calibration of hydrological models. Environ. Modell. Softw 31, 28–36.
Tang, Y., Reed, P.M., Kollat, J.B., 2007. Parallelization strategies for rapid and robust evolutionary multiobjective optimization in water resources applications. Adv. Water Resour. 30 (3), 335–353.
Tuppad, P., Douglas-Mankin, K.R., Lee, T., Srinivasan, R., Arnold, J.G., 2011. Soil and water assessment tool (SWAT) hydrologic/water quality model: extended capability and wider adoption. Trans. ASABE 54 (5), 1677–1684.
U.S. Environmental Protection Agency (US EPA), 2008. Handbook for developing watershed plans to restore and protect our waters. Report no. 841-B-08-002. Office of Water, Washington, DC, USA.
Vivoni, E.R., Mascaro, G., Mniszewski, S., Fasel, P., Springer, E.P., Ivanov, V.Y., Bras, R. L., 2011. Real-world hydrologic assessment of a fully-distributed hydrological model in a parallel computing environment. J. Hydrol. 409 (1–2), 483–496.
Vrugt, J.A., O Nuallain, B., Robinson, B.A., Bouten, W., Dekker, S.C., Sloot, P.M.A., 2006. Application of parallel computing to stochastic parameter estimation in environmental models. Comput. Geosci. 32 (8), 1139–1155.
Wai, O.W.H., Lu, Q.M., 2000. An efficient parallel model for coastal transport process simulation. Adv. Water Resour. 23 (7), 747–764.
Wood, E.F., Roundy, J.K., Troy, T.J., van Beek, L.P.H., Bierkens, M.F.P., Blyth, E., de Roo, A., Döll, P., Ek, M., Famiglietti, J., Gochis, D., van de Giesen, N., Houser, P., Jaffé, P. R., Kollet, S., Lehner, B., Lettenmaier, D.P., Peters-Lidard, C., Sivapalan, M., Sheffield, J., Wade, A., Whitehead, P., 2011. Hyperresolution global land surface modeling: meeting a grand challenge for monitoring Earth's terrestrial water. Water Resour. Res. 47 (5), W05301.
Wu, Y.P., Li, T.J., Sun, L.Q., Chen, J., 2013. Parallelization of a hydrological model using the message passing interface. Environ. Modell. Softw. 43, 124–132.
Wu, Y.S., Zhang, K.N., Ding, C., Pruess, K., Elmroth, E., Bodvarsson, G.S., 2002. An efficient parallel-computing method for modeling nonisothermal multiphase flow and multicomponent transport in porous and fractured media. Adv. Water Resour. 25 (3), 243–261.
Yeh, G.T., Shih, D.S., Cheng, J.R.C., 2011. An integrated media, integrated processes watershed model. Comput. Fluids 45 (1), 2–13.
Zhang, X., Izaurralde, R.C., Zong, Z., Zhao, K., Thomson, A.M., 2012. Evaluating the efficiency of a multi-core aware multi-objective optimization tool for calibrating the SWAT model. Trans. ASABE 55 (5), 1723–1731.
Zhang, X.S., Beeson, P., Link, R., Manowitz, D., Izaurralde, R.C., Sadeghi, A., Thomson, A.M., Sahajpal, R., Srinivasan, R., Arnold, J.G., 2013. Efficient multi-objective calibration of a computationally intensive hydrologic model with parallel computing software in Python. Environ. Modell. Softw. 46, 208–218.