MASTER IN HIGH PERFORMANCE COMPUTING

Reengineering and optimization of GEOtop software package

Supervisors:
Giacomo BERTOLDI,
Alberto SARTORI,
Stefano COZZINI

Candidate:
Elisa BORTOLI

4th EDITION
2017–2018

Acknowledgements

The research reported in this work was supported by OGS, CINECA and EURAC Research under HPC-TRES program award number 2017-20.

The computational results presented have been achieved [in part] using the Vienna Scientific Cluster (VSC).

For the test cases, data from the Long Term Ecological Research Area Mazia Valley (South Tyrol, Italy) have been used.

Giacomo Bertoldi, Alberto Sartori and Stefano Cozzini are acknowledged for the thesis supervision.

Siegfried Höfinger, Samuel Senorer, Martin Palma, Luca Cattani, Christian Brida and Emanuele Cordano are acknowledged for their technical support.

Contents

List of Figures

List of Tables

List of Abbreviations

DEM	**D**igital **E**levation **M**odel
IHSS	**I**ntegrated **H**ydrologic **S**urface **S**ubsurface
LSM	**L**and **S**urface **M**odels
LTER	**L**ong **T**erm **E**cological **R**esearch **S**ite
OOP	**O**bject **O**riented **A**pproach
RAII	**R**esource **A**cquisition **I**s **I**nitialization

Chapter 1

Introduction

The GEOtop hydrological scientific package is an integrated hydrological model that simulates the heat and water budgets at and below the soil surface (Rigon, Bertoldi, and Over, 2006). It describes the three-dimensional water flow in the soil and the energy exchange with the atmosphere, considering the radiative and turbulent fluxes. Furthermore, it reproduces soil freezing and thawing processes, and it simulates the temporal evolution of snow cover, soil temperature and moisture. The model can be applied both at the plot and the catchment scale to study the long term water budget and runoff production. The model has been applied to a variety of scientific problems, ranging from estimation of runoff and water budget in small - medium chatchments ($< 1000\ m^2$), studies related to the water-soil-vegetation interactions, snow cover in mountain areas, climate change impact assessment (for a full reference list see `http://geotopmodel.github.io/geotop/materials/publication-list.html`). One version of the model is currently used in an operational snow forecasting system (`http://www.mysnowmaps.com`).
The core components of the package were presented in the 2.0 version (Endrizzi et al., 2014), which was released as Free Software Open-source project under GNU General Public License v3.0. The code was written in C language. However, despite the high scientific quality of the project, a modern software engineering approach was still missing. Such weakness hindered its computational efficiency, its scientific potential and its use both as a standalone package and, more importantly, in an integrated way with other hydrological software tools and earth system models.
A poor engineering is typical issue of scientific softwares, whose goal is the creation of new scientific knowledge; the emphasis placed on software quality (i.e., correctness of code, maintainability, and reliability) has been historically lower than seen in more traditional software engineering (Heaton and Carver, 2015). More in general, the scientific software community is facing a crisis created by the confluence of disruptive changes in computing architectures and new opportunities for greatly improved data availability a simulation capabilities (See the scheme in Fig.1 taken from Ideas Productivity project). There is therefore the need, in order to keep productive well established scientific softwares to perform a software refactoring to develop efficient codes for parallel architecture. A suitable test case is the GEOtop model, an integrated hydrological model which started to be developed in 2000, and, since them, continuously evolved to address a number of scientific and applied problems, but also increasing it complexity.

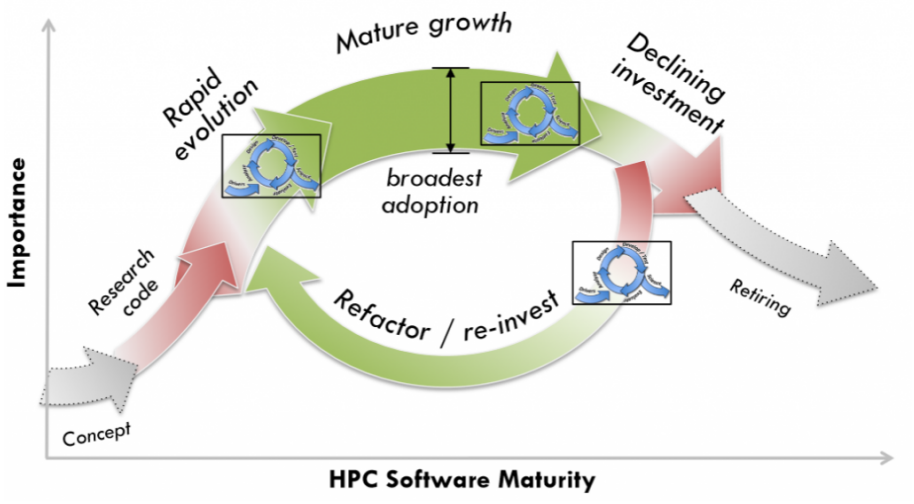


FIGURE 1.1: Schematic representation of the life cycle of a scientific software (from Ideas Productivity project).

The goal of this project is to perform a software re-engineering and refactoring of the GEOtop model code to create a robust and stable scientific software package, optimized for modern parallel clusters, open to the scientific community, easily usable by researchers and experts, and interoperable with other packages. Specifically, this thesis aims to:

- restructure the code from C to C++, taking advantage of an Object-Oriented Programming;
- clean the code, rewriting the old data structures;
- optimize the maths, replacing the computationally expensive operations with faster ones;
- parallelize the code with OpenMP, to decrease run time.

The thesis is structured as follows. First, will be given a brief overview of the model structure, code and numeric. Then, the software re-engineering work will be described in detail. In order to test model performance three representative experimental test cases have been selected among the large suite of possible models configurations. For the selected test cases, a code profiling has been performed. On the basis of those results a code optimization has been performed, improving the efficiency of most expensive mathematical operation and employing OpenMP parallelism for the thread-safe parts of the code. Then, performances and differences of the re engineered 3.0 code are compared with the original 2.0 version. Finally, future code developments towards a further code optimization are discussed.

Chapter 2

Model overview

GEOtop simulates the fluxes and budgets of energy and water on a landscape defined by three-dimensional grid boxes, whose surfaces come from a digital elevation model (DEM) and whose lower boundaries are located at some specified spatially varying depth, as shown in Fig. 2.1. Surface boundary conditions are given by hydrometeorological measurements (rainfall, temperature, wind velocity) (Bertoldi, 2004), regionalized with the approaches described in Liston and Elder, 2006 or in Bavay and Egger, 2014, depending on the code version. A general introduction on the model is given in Rigon, Bertoldi, and Over, 2006. The users manual can be found online here (Endrizzi et al., 2011). In this thesis only a brief overview will be given.

2.1 Landscape and equation discretization

GEOtop requires preprocessing of the catchment DEM to estimate drainage directions, slopes, curvature, the channel network structure, shadowing, and the sky view factor. Surface runoff is modeled to follow the terrain surface according to a so-called D8 topology as in Orlandini et al., 2003. The DEM identifies also the plan view of a three-dimensional grid on which all the model's equations are discretized. The grid cells are identified as hillslope or river network cells. River network cells are treated the same as hillslope cells except for the routing of surface runoff. For each cell, different land cover and soil properties could be defined.

2.2 Water and energy budgets

The system of equations representing the water balance in the soil is:

$$\frac{\partial \theta_w^{ph}}{\partial t} + \frac{\rho_i}{\rho_w}\frac{\partial \theta_i}{\partial t} = 0 \tag{2.1}$$

$$\frac{\partial \theta_w^{fl}}{\partial t} + \nabla \cdot (-K \nabla H) + S_w = 0 \tag{2.2}$$

where $d\theta^{ph}$ is the fraction of liquid water content in soil subject to phase change, $d\theta^{fl}$ is the fraction of liquid water content transferred by water flux,

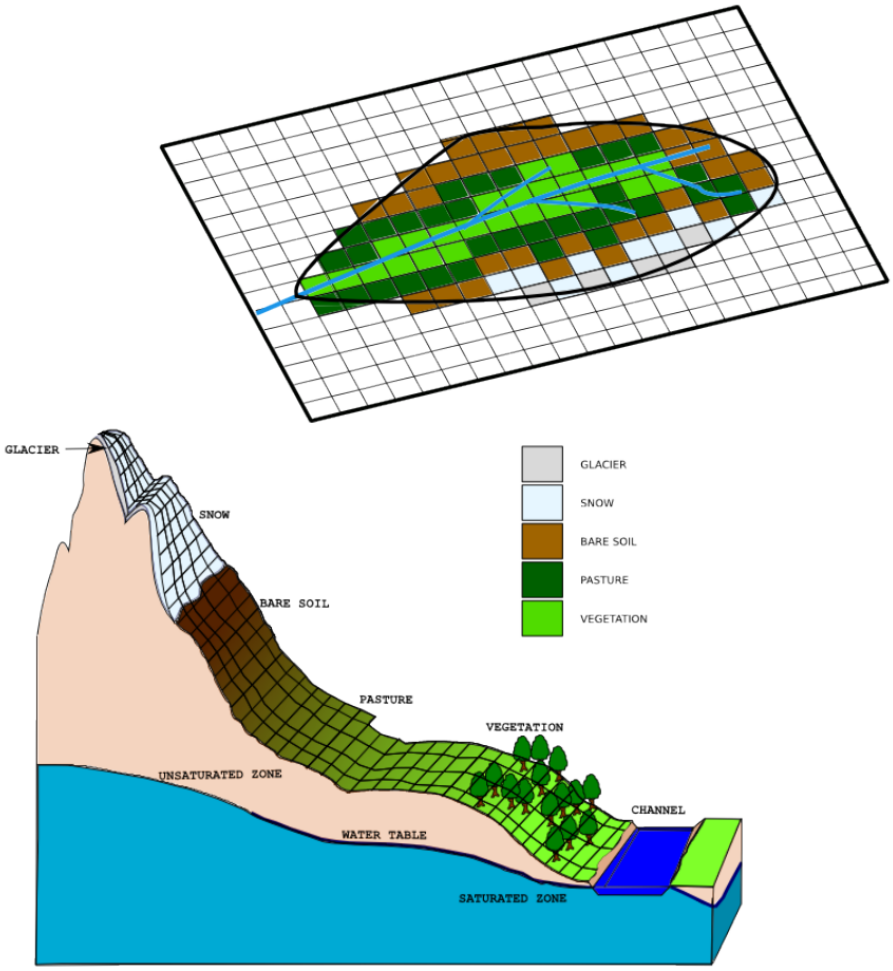


FIGURE 2.1: Classification of a slope surface in a mountain basin based on the land cover (from Endrizzi et al., 2011).

ρ_i is the density of ice, θ_i is the fraction of ice in soil, K is the hydraulic conductivity, H is the sum of the pressure and potential heads, S_w is the mass sink term.
The equation representing the energy balance in a soil volume subject to phase change is:

$$\frac{\partial U^{ph}}{\partial t} + \nabla \cdot G + S_{en} - \rho_w [L_f + c_w (T - T_{ref})] S_w = 0 \tag{2.3}$$

where U^{ph} is the volumetric internal energy of soil subject to phase change, t is time, $\nabla\cdot$ is the divergence operator, G is the heat conduction flux, S_{en} is the energy sink term, L_f is the latent heat of fusion, ρ_w is the density of liquid water in soil, T is the soil temperature and T_{ref} is the reference temperature at which the internal energy is calculated.

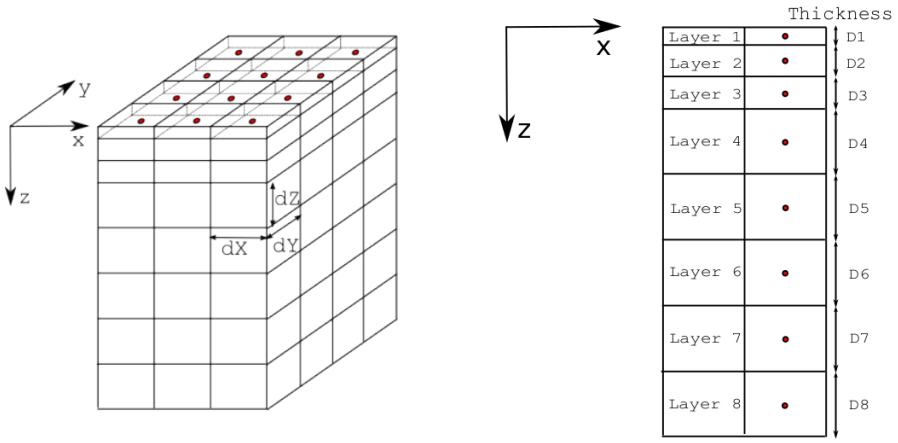


FIGURE 2.2: 3D calculation grid and discretization on the x-z plane; the red points, at the center of the cell, coincide with the calculation grid points (from Endrizzi et al., 2011).

2.3 Numerics

In this section is reported a synthesis of the GEOtop model numerical approach, taken from Endrizzi et al., 2017. In order to reduce the complexity of the numerical method, Eqs. (2.2) and (2.3) are linked in a time-lagged manner, instead of solving them in a fully coupled way. Both equations have the same form, which can be generalized as:

$$\frac{\partial F(\kappa)}{\partial t} + \nabla \cdot (-\kappa(\chi)\nabla\chi) + S = 0 \tag{2.4}$$

where χ is the unknown function of space and time, F a non-linear function of the unknown, S is the sink term and κ is a conductivity function of the unknown.
All the derivates are discretised as finite differences. Therefore the following relation is obtained.

$$\frac{F(\chi_i^{n+1}) - F(\chi_i^n)}{\Delta t} - \sum_j^M \frac{\kappa_{ij}^m}{D_{ij}}(\chi_j^m - \chi_i^m) + S_i = G_i \tag{2.5}$$

where the equation is written for the generic $i-th$ cell; n represents the previous time step (known solution), $n+1$ is the next time step (unknown solution), Δt is the time step, j is the index of the M adjacent cells with which the i cell can exchange fluxes, m represents a time instant between n and $n+1$, κ_{ij} is the conductivity between the cell i and j, D_{ij} is the distance between the centres of the cells i and j, S_i is the sink term and G_i is the residual that is to be minimized to find a solution.
Eq. (2.5) is a system of N equations and the second term on the left-hand side is the sum of the fluxes exchanged with the neighbouring cells. The variables at the instant m are represented with a linear combination between the instant n and $n+1$.

Several cases are possible:

- $m = n$: the method is fully explicit and unstable;
- $m = n + 1/2$: the method has a second order precision but might not be always stable;
- $m = n + 1$: the method has a first order precision but is unconditionally stable.

Since there are more concerns on stability than precision, the last is the chosen method.
A solution of Eq. (2.5) is sought with a special Newton-Raphson method, with the following sequence (Kelley, 2003):

$$\chi^{n+1} = \chi^n + \lambda_d \mathbf{d}(\chi^n) \tag{2.6}$$

where χ is the vector χ_i that appears in Eq. (2.5), $\mathbf{d}$ denotes the Newton direction and λ_d is the path length (a scalar, <= 1) found with a line searching method like the Armijo rule (Armijo, 1966). The quantity $\lambda_d \mathbf{d}(\chi^n)$ is also referred to as the Newton step.
The Newton direction is obtained solving the following linear system:

$$\mathbf{G}'(\chi^n)\mathbf{d} = -\mathbf{G}(\chi^n) \tag{2.7}$$

where $\mathbf{G}$ is the vector $\mathbf{G}_i$ that appears in Eq. (2.5) and $\mathbf{G}'(\chi^n)$ denotes the Jacobian matrix $G'(\chi^n) = \partial G_i(\chi)/\partial \chi_j$. If Eq. (2.4) is solved neglecting the lateral gradients, the number of adjacent cells that actually considered is maximun 2 (i.e. the cell below and above). Therefore the matrix $\mathbf{G}'(\chi^n)$ is tridiagonaland symmetric, and then invertible with simple direct methods (El-Mikkawy and Karawia, 2006).
On the other hand, if Eq. (2.4) is solved fully three-dimensionally, M can be up to 6 and therefore $\mathbf{G}'(\chi^n)$ is a symmetric and sparse matrix; its inversion is a more complex problem (Niessner, 1983). In this case the linear system in Eq. (2.7) is solved approximately with an iterative method, the BiCGSTAB Krylov linear solver (Van Der Vorst, 1992). This iterative process becomes an inner iteration, nested in the outer iteration defined in (2.6).

2.4 Software package

2.4.1 Simulation flow chart

The model transforms the input given by the user into results, by solving the energy and mass balance in the calculation domain (Endrizzi et al., 2017). As reported in Fig. 2.3 GEOtop does the following activities:

- **Read input data**. In this phase the model reads: (i) the keywords and parameters specified in the main configuration file called *geotop.inpts*; (ii) the topographic maps, as the DEM, the land cover map, and, if available, the maps with soil type, river drainage networks, the maps with the initial conditions; (iii) other optional parameters. If a parameter or a map is not specified with the proper keyword, it assumes a default value.

- **Create and initialize mesh**. It creates the calculation mesh according to the grid size of the land cover map and the vertical nodes spacing defined for the vertical grid. Then it initializes the temperature and water pressure head of each node with the initial conditions and sets the physical parameters according to what specified by the keywords.

- **Read meteo data**. During this phase, it incorporates the meteorological input data for each available meteorological station: these data represent the forcing that will drive the simulation, producing the dynamic boundary conditions for the surface nodes. Finally, GEOtop sets the initial simulation time to initialize the simulation counter: this will allow to compare the current simulation time with the expected simulation end time

At this point the time loop for the calculation and the printing routines begin. In particular, at each calculation time step, GEOtop fulfills the following tasks, as illustrated in Fig. 2.3:

- **Distribute meteorological forcing**. This allows to spatially distribute the meteorological forcing, measured in discrete meteo station, in all the calculation cells. This methodology is based on Liston and Elder, 2006, for the code version 2.0, or on the METEO-IO library (Bavay and Egger, 2014) for the code version 2.1.

- **Energy balance**. In this phase the energy balance equation is solved. This encompasses the calculation of the surface energy fluxes, the vegetation module, the snow/glacier module and the routine that the calculates the soil temperatures and ice content.

- **Water balance**. In this phase the mass balance equation is solved. This encompasses the calculation of the infiltration routine to determine the pore water pressure and water content through a 3D Richards solver. Eventually, the runoff and channel routing routines, based on a shallow-water solver, will allow to determine the discharge at the basin outlet.

- **Write output**. This phase is intended to print the point information and the maps according to the desired output frequency.
- **Update and check time**. This phase updates the time with the calculation time step and compares the new time with the simulation end time, to verify whether to stop the simulation or loop again. The model uses a dynamic calculation time step. If the convergence criteria is not reached either for the solution of the energy of the water budget, then the time step is reduced. If the current simulation time exceeds the end of the simulation, then the program stops and deallocates all the structures.

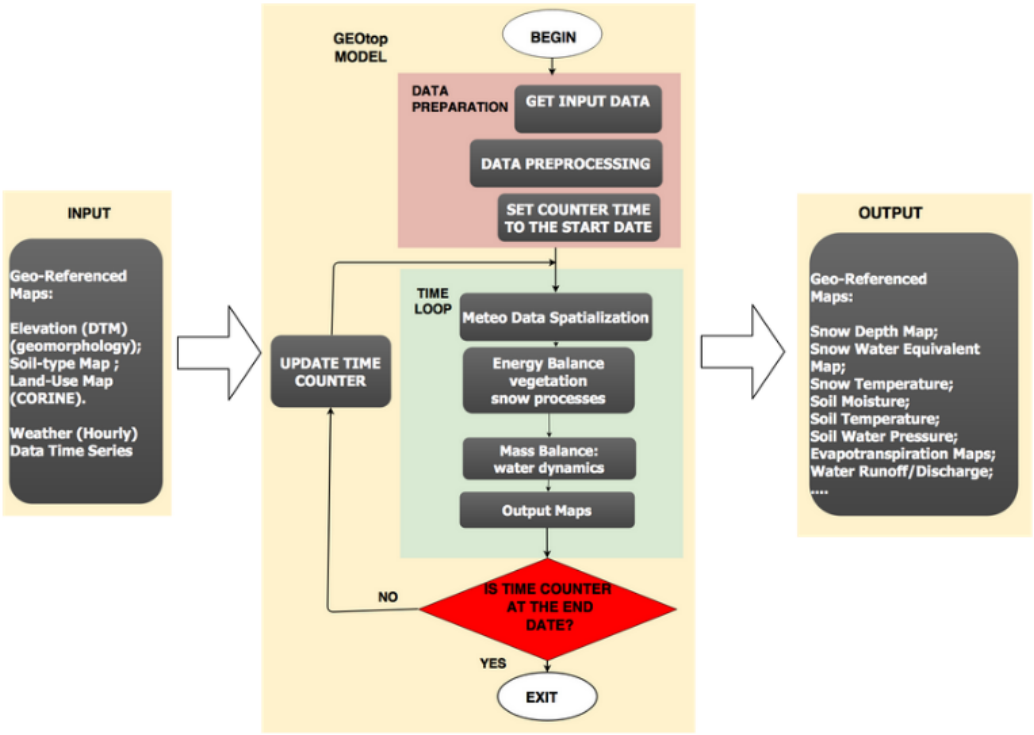


FIGURE 2.3: GEOtop flow chart: model point of view for accomplishing a simulation (Courtesy of E. Cordano).

In terms of model functions, the call structure is quite complex. The most relevant functions calls are illustrated in the scheme of Fig. 2.4, which has been obtained parsing the source code with Doxygen.

2.4.2 Simulation types

The model can run with two different domain configurations:

- **1D**: only vertical fluxes are considered, so mass and energy balance are performed at local scale. Actually some processes are mainly 1-dimensional (i.e., soil temperature and snow profiles), therefore they can be investigated using GEOtop in a simplified manner. In such a way the computational domain is reduced to one vertical column aligned to a Cartesian grid. Examples of processes mainly characterized by 1D-dynamics are vertical water infiltration, plot scale estimation of snow melt and vegetation processes.

- **3D**: both vertical and lateral fluxes are taken into account so balances are done at basin scale. Examples of processes mainly characterized by 3D-dynamics are atmosphere-vegetation interactions, groundwater movement, catchment scale water budgets. Usually this setup needs more calculations so it is more CPU-intensive.

The model can be also run turning off or on the main processes, which are the energy budget and the water budget calculation. For example, to simulate snow dynamics, only the energy budget is needed; to simulate water infiltration, only the water budget. To simulate in complete way catchment scale hydrological processes, a 3D calculation of both budgers is needed.
On a typical workstation, a full 3D one year simulation over a grid of about $200x200$ pixels could require between 6 and 24 hours of computatio time. For this reason, there is the need to optmize and parallelize the code in order to cope with modern scientific and operational needs.

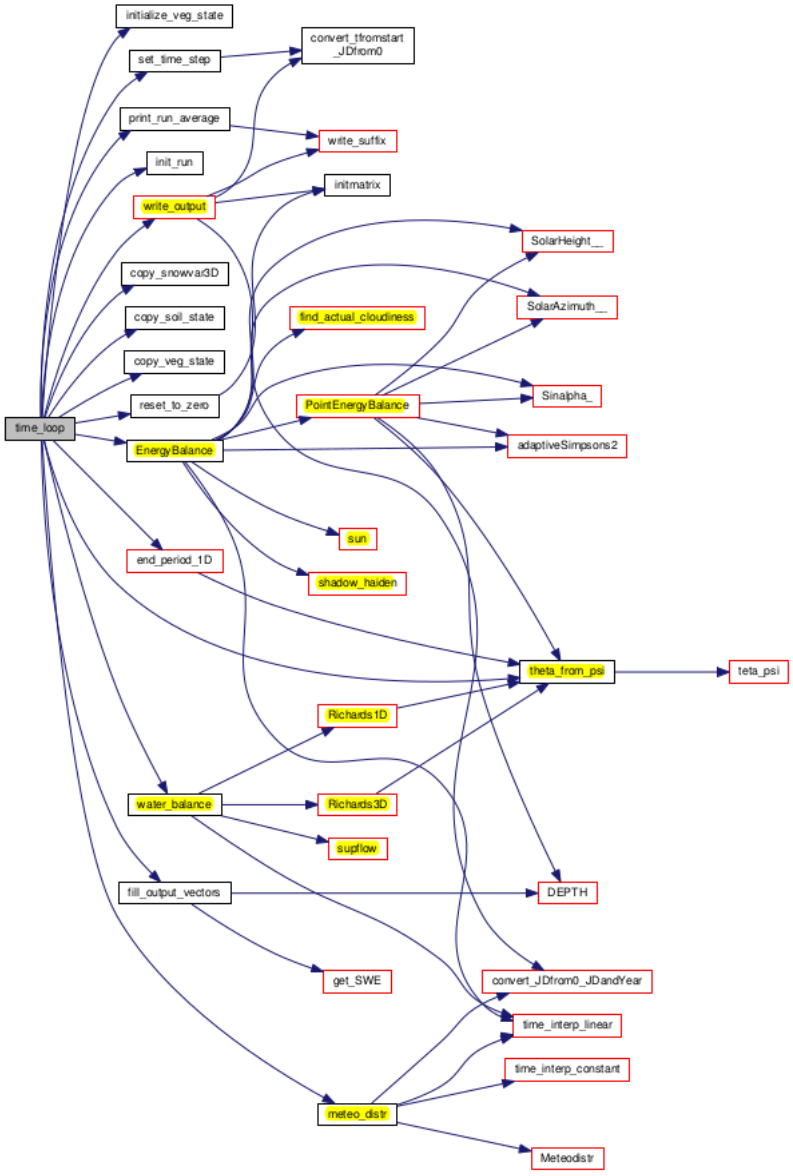


FIGURE 2.4: Most relevant functions calls of GEOtop derived from Doxygen. In yellow are underlined the functions linked with the main physical processes modelled by GEOtop.

Chapter 3

GEOtop 3.0

3.1 Background

The latest versions of GEOtop are:

- **2.0**: written in C, released in 2014 as free software open-source project, scientifically tested and published. This version is available on the github repository https://github.com/geotopmodel/geotop at branch **se27xx**. However, despite the high scientific quality of the project, a modern software engineering approach was still missing.

- **2.1**: developed from 2014, written in C++, open source and documented on the same github repository but at branch **master**. This version, differently from the 2.0, was developed from the beggining using the git version-control system, and Travis-CI (https://docs.travis-ci.com/) allowed to continuously check the correctness of the build over a wide number of tests cases.

  The main advantage of this new version is the possibility to use MeteoIO library (Bavay and Egger, 2014), that provides a uniform interface to meteorological data (https://models.slf.ch/p/meteoio/); unfortunately, the output results are different compared to the validated 2.0 and only a few people were working on the scientific validation. Besides, the code is neither modular nor flexible, and it is characterized by code repetitions and unsolved bugs, difficult to find.

Hence a new version was needed that had to be scientifically validated, easy to compile and run, modular and flexible, tested as much as possible and computationally efficient. The code development will have to fulfill the so called "best programming practices", a set of rules that have solid foundations in research and experience, and that improve scientists' productivity and their software reliability (Wilson et al., 2014). These will be referred and explained in details in the following sections, when describing the new features of GEOtop 3.0; the code package can be found in the same github repository at branch **v3.0**.

3.2 Easy to compile and run

Using a build system tool to automate workflows can avoid errors and inefficiencies from repeating commands manually (Wilson et al., 2014); this is one of the "best programming practices", and also a way to simplify the code compiling, running and debugging. Meson build system tools (https://mesonbuild.com/) was used since it is fast, allows for modularity and it can be easily coupled with gdb (https://www.gnu.org/software/gdb/). However, the usage of CMake (https://cmake.org/) was preserved to maintain backward compatibility.

3.3 Modular and flexible

C++ programming language was chosen because, in addition to the facilities provided by C (in which the scientifically validated GEOtop version was written), it provides flexible and efficient facilities for defining new types that closely match the concepts of the application: this technique for program construction is called *data abstraction*(Stroustrup, 2013).
These user-defined types are named *classes*: they are an expanded concept of data structures, containing a series of variables named *members* and a series of procedures named *methods*. An object of a class contains type information and it can be used in contexts in which its type cannot be determined at compile time; programs using objects of such types are often called *Object-based* or *Object-Oriented*. The advantages of OOP exploited in GEOtop 3.0 are the following: (http://www.c4learn.com/cplusplus/oop-advantages/):

(1) it provides a clear modular structure for programs which makes it good for defining abstract datatypes in which implementation details are hidden;

(2) objects can also be reused within and across applications, lowering the cost of development and decreasing potential mistakes;

(3) it makes software easier to maintain and modify, as new objects can be created with small differences to existing ones;

(4) it has the feature of memory management through RAII (Resource Acquisition Is Initialization) technique, which binds the life cycle of a resource (i.e., allocated heap memory) to the lifetime of an object (https://en.cppreference.com/w/cpp/language/raii), whose dedicated memory is allocated by a constructor and deallocated by a destructor, preventing memory leaks.

(5) it is suitable for large projects and fairly efficient.

Moreover, another advantage of C++ is the possibility to write code in a way that is independent of any particular type thanks to the usage of *templates*. A template is a blueprint or formula for creating a generic *class* or a function, than can be used with different data types (https://www.tutorialspoint.

com/cplusplus/cpp_templates.htm): this avoids unnecessary code repetitions and reduce potential mistakes, without penalties at runtime.
Additionally, C++ has a very rich function libraries, designed for portability.
All these new aspects and features, some of which could be considered "best programming practices" (i.e., (1) makes the code easy to understand and (3) allows for code reusage) were exploited in the new data structures:

- Vector<T>, Matrix<T> and Tensor<T> (whose some parts are reported in code boxes 3.4 and 3.5), used to conceptually define vectors, matrices and tensors;
- RowView<T> and MatrixView<T>, used to respectively access a row from an object of type Matrix<T> and access a matrix from an object of type Tensor<T>.

These new classes were defined in a future perspective to allow a rewriting of the **linear algebra** and a **parallelization**, not easily achievable using the old data structures defined using the fluid turtle library (http://www.ing.unitn.it/~rigon/FLUIDTURTLE/LIBRARIES/BASICS/).
Now, differently from GEOtop 2.0, the data structures can be accessed in a uniform way by the user thanks to the operator overloading, and, in order to prevent mistakes, only the interfaces are exposed, while the implementation details (i.e., private class members) are hidden from outside of the class, according to data-hiding technique (1).
Moreover in the new data structures some "special" operators were defined not only to simply access an element of the structure (i.e., []) but also to perform a bound check (i.e. ()) as it will be explained in section 3.4.
Actually, in GEOtop 2.0, the element indexing was not equal for all the variables, whose first index could be 0 (like C/C++) or 1 (like Fortran); the latter was the default choice (code boxes 3.2 and 3.3) so in the former case another allocation function was used (code box 3.1). This error-prone technique led to segmentation fault for some tests; the issues were solved, during the debug phase, thanks exactly to the development of the () operator.

```
DOUBLEVECTOR *new_doublevector0(long nh)
{
  DOUBLEVECTOR *m;
  m=(DOUBLEVECTOR *)malloc(sizeof(DOUBLEVECTOR));
  if (!m) t_error("allocation failure in DOUBLEVECTOR()");
  m->isdynamic=isDynamic;
  m->nl=0;
  m->nh=nh;
  m->co=dvector(m->nl,nh);
  return m;
}
```

LISTING 3.1: Vector allocation for GEOtop 2.0 using the first index equal to 0 (alloc.c)

```
#define NL 1 /* Numerical Recipes allocation routines allow to have
                arbitrary  subscripts for vector and matrixes.
                The fluid turtle library restrict this freedom by
    setting
                their  lower value to NL  */
```

LISTING 3.2: Definition of the lower bound for a data structure in GEOtop 3.0 (turtle.h)

```
DOUBLEVECTOR *new_doublevector(long nh)
{
  DOUBLEVECTOR *m;
  m=(DOUBLEVECTOR *)malloc(sizeof(DOUBLEVECTOR));
  if (!m) t_error("allocation failure in DOUBLEVECTOR()");
  m->isdynamic=isDynamic;
  m->nl=NL;
  m->nh=nh;
  m->co=dvector(m->nl,nh);
  return m;
}
```

LISTING 3.3: Vector allocation for GEOtop 2.0 using the first index equal to 1 (alloc.c)

```
#include "matrix.h"
#include "matrixview.h"

template <class T> class RowView;
template <class T> class MatrixView;

template <class T> class Tensor {

public:
...
    /** Given a layer (k) and a row (i), it gives all the elements
    ** corrisponding to different columns */
    RowView<T> row(const std::size_t k, const std::size_t i) {
        GEO_ASSERT_IN_RANGE(k, ndl, ndh);
        GEO_ASSERT_IN_RANGE(i, nrl, nrh);
        return RowView<T> { &co[(i-nrl)*n_col + (k-ndl)*(n_row*
    n_col)], nch, ncl};
    };

    MatrixView<T> matrix(const std::size_t k) {
        GEO_ASSERT_IN_RANGE(k, ndl, ndh);
        return MatrixView<T> { &co[(k-ndl)*(n_row*n_col)], nrh, nrl
    , nch, ncl};
    };
```

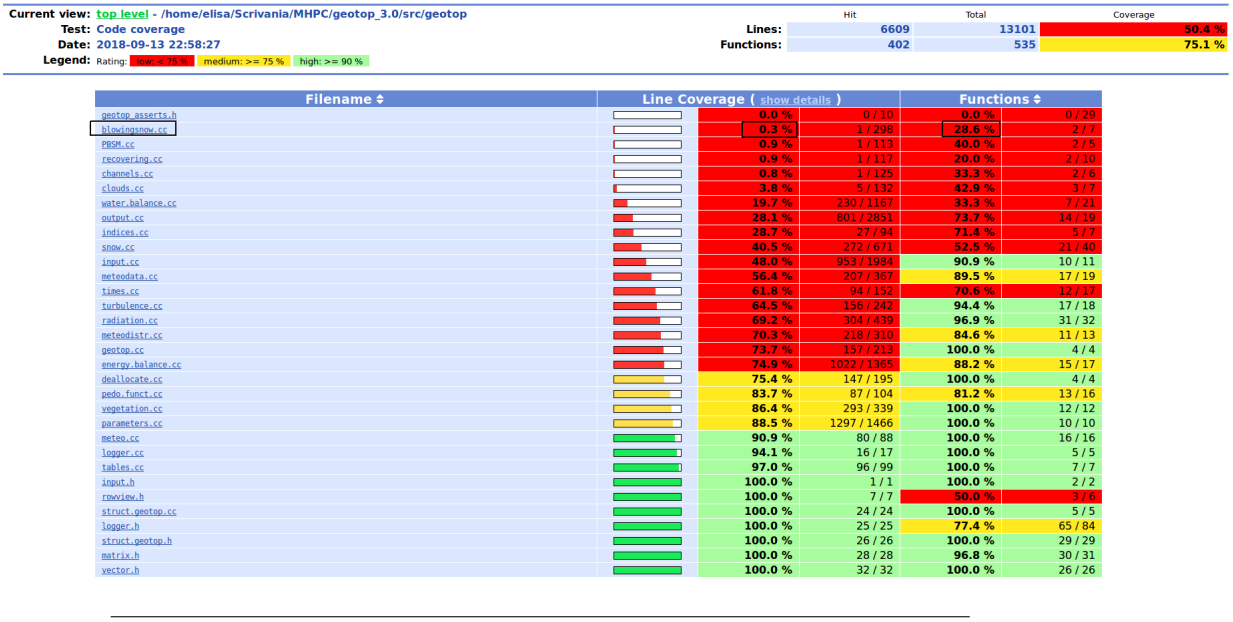
LISTING 3.4: Part of the header file tensor.h of GEOtop 3.0 showing modularity (1) and code reusage (2)

```
 * constructor
 * @param _nrl,_nrh lower and upper bound for rows
 * @param _ncl,_nch lower and upper bound for columns
 * @param _ndl,_ndh lower and upper bound for depth
 */

  Tensor(const std::size_t _ndh,  const std::size_t _ndl,
         const std::size_t _nrh, const std::size_t _nrl,
         const std::size_t _nch,  const std::size_t _ncl):
  ndh{_ndh}, ndl{_ndl}, nrh{_nrh}, nrl{_nrl}, nch{_nch}, ncl{_ncl
 },
  n_dep{ndh-ndl+1}, n_row{nrh-nrl+1}, n_col{nch-ncl+1},
  co { new T[(ndh-ndl+1)*(nrh-nrl+1)*(nch-ncl+1)]{} } {} // init
 to 0

  Tensor(const std::size_t d, const std::size_t r, const std::
 size_t c):
  Tensor{d,1,r,1,c,1} {}

  /** destructor. default is fine */
  ~Tensor() = default;
```

LISTING 3.5: Part of the header file tensor.h. of GEOtop 3.0 showing the application of RAII concept (4)

3.4 Tested as much as possible

Coverage analysis of a program can be a significant component in confident assessment of overall software quality since it gives a clear measure of code testing (Horgan, London, and Lyu, 1994), allowing to discover its untested parts.
Gcov coverage testing tool (https://gcc.gnu.org/onlinedocs/gcc/Gcov.html) together with Lcov graphical front-end, were used to find what lines of code were actually executed. Actually Lcov collects gcov data for multiple source files and creates HTML pages containing the source code annotated with coverage information, also adding overview pages for easy navigation within the file structure (http://ltp.sourceforge.net/coverage/lcov.php).
Analyzing the code coverage for all the 1D tests (Fig. 3.1) it was noticed that many lines and functions were not used; for example in the source file blowingsnow.cc, dealing with snow transport and deposition, < 30% of functions were used: this could happen because it was not snowing or because some functions should have been used but they were not.



Current view: top level - /home/elisa/Scrivania/MHPC/geotop_3.0/src/geotop

Test: Code coverage

Date: 2018-09-13 22:58:27

Legend: Rating: low: < 75 % medium: >= 75 % high: >= 90 %

	Hit	Total	Coverage
Lines:	6609	13101	50.4 %
Functions:	402	535	75.1 %

Filename	Line Coverage (show details)		Functions	
geotop_asserts.h	0.0 %	0 / 10	0.0 %	0 / 29
blowingsnow.cc	0.3 %	1 / 298	28.6 %	2 / 7
PBSM.cc	0.9 %	1 / 113	40.0 %	2 / 5
recovering.cc	0.9 %	1 / 117	20.0 %	2 / 10
channels.cc	0.8 %	1 / 125	33.3 %	2 / 6
clouds.cc	3.8 %	5 / 132	42.9 %	3 / 7
water_balance.cc	19.7 %	230 / 1167	33.3 %	7 / 21
output.cc	28.1 %	801 / 2851	73.7 %	14 / 19
indices.cc	28.7 %	27 / 94	71.4 %	5 / 7
snow.cc	40.5 %	272 / 671	52.5 %	21 / 40
input.cc	48.0 %	953 / 1984	90.9 %	10 / 11
meteodata.cc	56.4 %	207 / 367	89.5 %	17 / 19
times.cc	61.8 %	94 / 152	70.6 %	12 / 17
turbulence.cc	64.5 %	156 / 242	94.4 %	17 / 18
radiation.cc	69.2 %	304 / 439	96.9 %	31 / 32
meteodistr.cc	70.3 %	218 / 310	84.6 %	11 / 13
geotop.cc	73.7 %	157 / 213	100.0 %	4 / 4
energy_balance.cc	74.9 %	1022 / 1365	88.2 %	15 / 17
deallocate.cc	75.4 %	147 / 195	100.0 %	4 / 4
pedo_funct.cc	83.7 %	87 / 104	81.2 %	13 / 16
vegetation.cc	86.4 %	293 / 339	100.0 %	12 / 12
parameters.cc	88.5 %	1297 / 1466	100.0 %	10 / 10
meteo.cc	90.9 %	80 / 88	100.0 %	16 / 16
logger.cc	94.1 %	16 / 17	100.0 %	5 / 5
tables.cc	97.0 %	96 / 99	100.0 %	7 / 7
input.h	100.0 %	1 / 1	100.0 %	2 / 2
rowview.h	100.0 %	7 / 7	50.0 %	3 / 6
struct.geotop.cc	100.0 %	24 / 24	100.0 %	5 / 5
logger.h	100.0 %	25 / 25	77.4 %	65 / 84
struct.geotop.h	100.0 %	26 / 26	100.0 %	29 / 29
matrix.h	100.0 %	28 / 28	96.8 %	30 / 31
vector.h	100.0 %	32 / 32	100.0 %	26 / 26

FIGURE 3.1: Code coverage for 1D tests using gcov.

In order to improve code reliability, a testing suite was set, comprising:

(1) 1D and 3D short test cases, consisting in the check of effective run of the executable with the all the inputs of 1D and 3D cases provided in the git repository, plus a comparison of output results between the 3.0 and 2.0 version, making the test passing if the absolute and relative differences were $< 10^{-5}$;

(2) unit tests for all the new added code (i.e. constructors, initialization and functions of the new data structures) using the Google test framework (`https://github.com/google/googletest`);

(3) bound checking when accessing elements of the new data structures; this is possible thanks to the access operator () that performs a range check when the code is compiled in debug mode.

For istance, to run all the 1D and 3D short test cases (1) the command that has to be typed in the build folder is:

```
meson test --suite geotop:1D --suite geotop:3D
```

and the output for every test will show the test name (i.e., Bro) and the simulation type (i.e., 1D) (see Tab. 3.1).

TABLE 3.1: Output of Meson test for a 1D short test case.

```
1/2 geotop:1D+Bro / 1D/Bro                  OK       11.13 s
2/2 geotop:1D+Bro / 1D/Bro.test_runner      OK        3.48 s
```

Examples of unit tests (2) are provided in the code boxes 3.6 and 3.7 where the correctness check is done in the expression `EXPECT_EQ`, performing a comparison between the first value, provided by the developer (since he/she already knows it for a simple case) and the second, calculated using the newly written code; if the two values are different, a statement error will be printed with both, allowing the developer to understand what happened.

```
TEST(Matrix, constructor_2args){
  Matrix<int> m{3,5}; // 3x5 matrix
  EXPECT_EQ( std::size_t{3}, m.n_row );
  EXPECT_EQ( std::size_t{5}, m.n_col );
}
```

LISTING 3.6: Unit tests for the constructor of a Matrix<T>.

```
TEST(Matrix, initialization){
  Matrix<int> m{2,2}; // 2x2 matrix
  EXPECT_EQ( 0, m(1,1) );
  EXPECT_EQ( 0, m(1,2) );
  EXPECT_EQ( 0, m(2,1) );
  EXPECT_EQ( 0, m(2,2) );

  #ifndef NDEBUG
  EXPECT_ANY_THROW( m(0,0));
  EXPECT_ANY_THROW( m(3,3));
 #else
  EXPECT_NO_THROW( m(0,0) );
  EXPECT_NO_THROW( m(3,3) );
  #endif
}
```

LISTING 3.7: Unit tests for the initialization of a Matrix<T>.

Examples of the access operator (3) for the class Vector was developed as shown in the code box 3.8 and 3.9; it can be noticed that the operator (), when the compiling mode is:

- RELEASE: it returns just the element, recalling the operator [];
- DEBUG: it calls the range-checked access operator at, which uses the macro `GEO_ERROR_IN_RANGE`, checking that the accessed index (i) is betwen the lower (nl) and upper (nh) bound of the vector.

```
  /** range-checked access operator */
   T &at(const std::size_t i) {
       GEO_ERROR_IN_RANGE(i, nl, nh);
       return (*this)[i];
   }
```

LISTING 3.8: Definition of the range-checked access operator for Vector<T>.

```
    T &operator()(const std::size_t i)
#ifdef NDEBUG
    noexcept
#endif
    {
#ifndef NDEBUG
        return at(i);
#else
        return (*this)[i];
#endif
    }
```

LISTING 3.9: Definition of the access operator for Vector<T>.

3.5 Computationally efficient

Several optimizations were implemented: all of them can be activated by a flag (as explained in Chapter 8) except one, that is the inline of all the functions in the previous existing source files pedo.func.cc, containing pedotransfer functions and statistic functions, and util_math.cc, containing mathematical rules to solve a linear system.
Inline function is an optimization technique used by the compilers especially to reduce the execution time (http://www.cplusplus.com/articles/2LywvCM9/). When the compiler inline-expands a function call, the function's code gets inserted into the caller's code stream: this can improve performance, because the optimizer can procedurally integrate the called code (https://isocpp.org/wiki/faq/inline-functions).
Since the functions in pedo.func.cc and util_math.cc are very much used and are called thousands of time during a test run (see Chapter 7), they were put in the correspondent header files pedo.func.h and util_math.h preceded by the keywords *inline*. Examples of inlined functions are reported in the code boxes 3.10 and 3.11.

```
inline double theta_from_psi(double psi, double ice, long l,
    MatrixView<double> &&pa, double pmin)
{

  const double s = pa(jsat,l);
  const double res = pa(jres,l);
  const double a = pa(ja,l);
  const double n = pa(jns,l);
  const double m = 1.-1./n;
  const double Ss = pa(jss,l);

  return teta_psi(psi, ice, s, res, a, n, m, pmin, Ss);

}
```

LISTING 3.10: Function inline of theta_from_psi in pedo.func.h.

```
inline double adaptiveSimpsons2(double (*f)(double x, void *p),
                          void *arg,   // ptr to function
                          double a, double b,  // interval [a,b]
                          double epsilon,  // error tolerance
                          int maxRecursionDepth)     // recursion
    cap
{
    double c = (a + b)/2, h = b - a;
    double fa = f(a, arg), fb = f(b, arg), fc = f(c, arg);
    double S = (h/6)*(fa + 4*fc + fb);
    return adaptiveSimpsonsAux2(f, arg, a, b, epsilon, S, fa, fb,
    fc,
                                maxRecursionDepth);
}
```

LISTING 3.11: Function inline of adaptiveSimpsons2 in util_math.h.

Chapter 4

Test cases

One of the major challenges in testing the GEOtop model code is related to the fact that this kind of integrated models can be used in a very wide range of operational conditions and spatial scales. Very different environments and climatic conditions can be considered. Addressed scientific topics range from the classical hydrologicals ones (i.e. runoff prediction) to ecological ones (i.e Evapotranspiration estimation), mountain cryospehere (i.e. snow processes). Typical model's applications are climate change impacts or risks assessments, as floods or droughts or landslides instability problems. This implies that it is quite challenging to test all the parts of the code. Moreover, the performances and the relative use of the different parts of code depend on the specific process considered.
In the GEOtop v.3 version a suite of more than 10 1D and 20 3D test cases are considered (https://github.com/geotopmodel/geotop/tree/v3.0/tests/), to cover the wide range of scientific problems than can be addressed by the GEOtop model.
However, for in this thesis three test cases are analyzed: one 1D case, **Matsch_B2_Ref_007**, representative of a full calculation of the water and energy budget at local scale, and two 3D cases, **snow_dstr_SENSITIVITY**, with only the energy budget and snow processes in winter and **Muntatschini_ref_005**, with both water and energy budget in summer. The three test cases were run for both a short and long time period for a detailed profiling and to check the optimization improvements on both short and long runs.

4.1 Matsch_B2_Ref_007

The test involves a hydro-meteorological stations named B2 and located in Montacini, a Long Term Ecological Research site (LTER) of the Mazia Valley (South Tyrol, Italy). The station is located at 1480 m a.s.l. in a mountain meadow area and it is characterized by a sandy loam soil (Bortoli, 2017). The GEOtop model has been already applied and scientifically validated against field observations in the work of Della Chiesa et al., 2014 and of Bortoli, 2017. The model has been employed in 1D mode activating the simulation of both water and energy budgets. The simulated time is:

- **1 month** for the short test (02/10/2009 00:00 - 02/11/2009 00:00);
- **~5 years** for the long test (02/10/2009 00:00 - 31/12/2015 23:00).

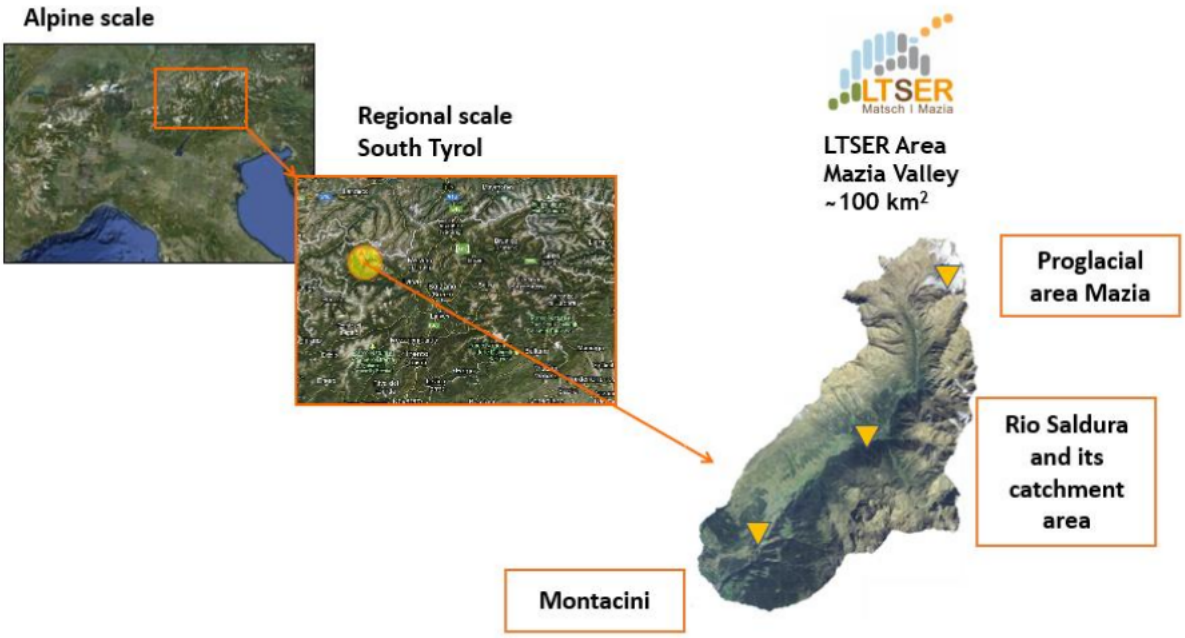


FIGURE 4.1: Area surrounding station B2: large view (from Bortoli, 2017)

.

4.2 snow_dstr_SENSITIVITY

The test is about the simulation of snow processes in the upper Saldura catchment, a small high-elevation catchment which is also part of the LTER Mazia, located in the upper Venosta valley (South Tyrol, Italy) (Engel et al., 2017). The input meteorological stations, indicated in Fig.4.2, are seven: B1, B2, B3, M3, M4, belonging to EURAC in the framework of the LTER, and Teufelsegg and Grawand, operated by the Hydrographic Office of the Autonomous Province of Bolzano. The GEOtop model has been scientifically validated against field observations in the work of Engel et al., 2017.
The model has been employed in 3D mode activating the simulation of only the energy budget.
The study area is 61 km^2 and the set cells are 10'140. The simulated time is:

- **1 day** for the short test (26/03/2010 17:00 - 27/03/2010 17:00);
- **1 month** for the long test (26/03/2010 17:00 - 26/04/2010 17:00).

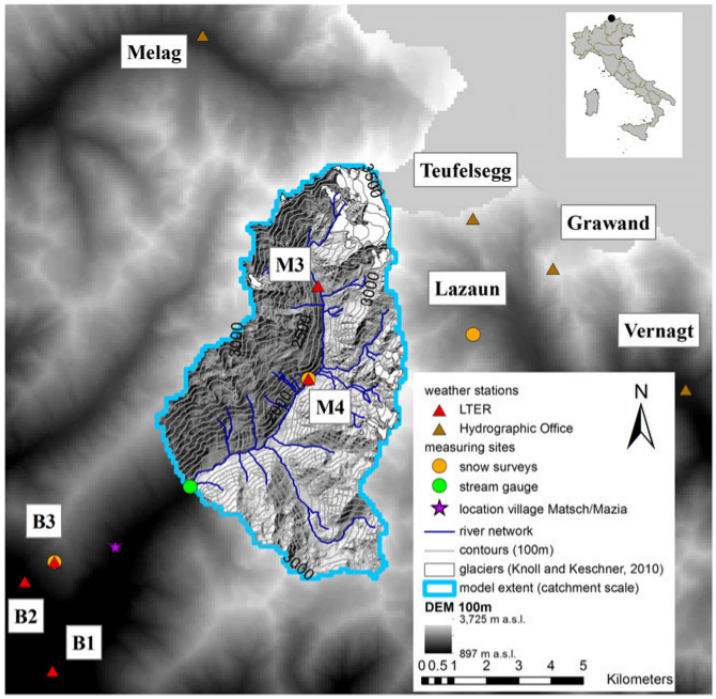


FIGURE 4.2: Analyzed area for snow test case (from Engel et al., 2017).

4.3 Muntatschini_ref_005

The study area in this test is Montacini (Figure 4.3): it is characterized by elevations between 900 and 2200 m a.s.l. and the main land covers are meadows and pastures. The input meteorological stations are four: B1, B2, B3, P2, all of them are LTER sites. The GEOtop model has been already applied and scientifically validated against field observations in the work of Della Chiesa et al., 2014 and of Bortoli, 2017.
The study area is ~4 km^2 and the set cells are 15'600. The simulated time is:

- **1 day** for the short test (26/03/2010 17:00 - 03/10/2009 00:00);
- **1 week** for the long test (02/10/2009 00:00 - 09/10/2009 00:00).

FIGURE 4.3: View of Montacini study site.

Chapter 5

Used architectures

5.1 Local pc

My local pc was used for profiling and for checking eventual improvements of the optimization. The CPU and cache characteristics are reported in Tab. 5.1; the total RAM is 16 GB. From now it will be referred with the name of CPU line *Intel Core*.

TABLE 5.1: CPU specifications of my local pc.

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	8
On-line CPU(s) list:	0-7
Thread(s) per core:	2
Core(s) per socket:	4
Socket(s):	1
NUMA node(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	94
Model name:	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
Stepping:	3
CPU MHz:	800.007
CPU max MHz:	3500,0000
CPU min MHz:	800,0000
BogoMIPS:	5183.87
Virtualization:	VT-x
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	6144K
NUMA node0 CPU(s):	0-7

5.2 VSC-3

The VSC-3 is an HPC system that was installed in summer 2014 at the Arsenal TU building (Objekt 214) in Vienna by Opens external link in new window-ClusterVision (http://vsc.ac.at/systems/vsc-3/).
It consists of 2020 nodes, each equipped with 2 processors belonging to the Ivy Bridge-EP family and internally connected with an Intel QDR-80 dual-link high-speed InfiniBand fabric.
VSC-3 was used to measure optimization improvements for long tests.
Compiling and running were done specifically on two nodes: n22-029 and n23-030, having the same specifics reported in Tab. 5.2; the total RAM is 128 GB. From now it will be referred with the name of CPU line *Intel Xeon*.

TABLE 5.2: CPU specifications of the used node of VSC-3.

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	32
On-line CPU(s) list:	0-31
Thread(s) per core:	2
Core(s) per socket:	8
Socket(s):	2
NUMA node(s):	2
Vendor ID:	GenuineIntel
CPU family:	6
Model:	62
Model name:	Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
Stepping:	4
CPU MHz:	1199.960
CPU max MHz:	3400,0000
CPU min MHz:	1200,0000
BogoMIPS:	5200.24
Virtualization:	VT-x
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	20480K
NUMA node0 CPU(s):	0-7,16-23
NUMA node1 CPU(s):	8-15,24-31

Chapter 6

Profiling

In order to find and track performance bottlenecks, the code was profiled with:

(1) likwid-perctr, that counts hardware performance events (https://github.com/RRZE-HPC/likwid/wiki/likwid-perfctr)

(2) callgrind, that records the call history among functions in the program's run as a call-graph (http://valgrind.org/docs/manual/cl-manual.html); this was used together with KCachegrind, a profile data visualization (https://kcachegrind.github.io/html/Home.html).

Moreover, a class *Timer*(3) was implemeted with which it was possible to measure the number of calls and time of some specific functions.
The profiling using (1) and (2) was done for both GEOtop versions 2.0 and 3.0 but considering only **short tests** due to the profiling overhead.
The analysis using (3) were performed only for GEOtop 3.0, since it involves the addition of some lines of C++ code, but both **short** and **long** tests were done.
All the profiling tests were performed only on Intel Core.

6.1 Likwid-perfctr

The analyzed groups during the runs for the **short** tests were:

- CYCLE_ACTIVITY, that measures cycle activities, giving an idea of the stalls caused by data traffic in the cache hierarchy;
- L2CACHE, that measures L2 cache miss rate/ratio, telling how many of the memory references required a cache line to be loaded from a higher level;
- L3CACHE, that measures L3 cache miss/ratio.

The typed commands was:

```
likwid-perfctr -g CYCLE_ACTIVITY -C 0 -g L2CACHE -g L3CACHE ./geotop TEST_DIR
```

The output of the first run for **B2** test case and GEOtop 3.0 is reported in Tabs. 6.1 and 6.2.

TABLE 6.1: Cycle activity for B2 test case.

Group 1: CYCLE_ACTIVITY

Event	Counter	Core 0
INSTR_RETIRED_ANY	FIXC0	13075903238
CPU_CLK_UNHALTED_CORE	FIXC1	6804707416
CPU_CLK_UNHALTED_REF	FIXC2	5092016616
CYCLE_ACTIVITY_STALLS_L2_PENDING	PMC0	24212141
CYCLE_ACTIVITY_STALLS_LDM_PENDING	PMC1	170785944
CYCLE_ACTIVITY_STALLS_L1D_PENDING	PMC2	29946993
CYCLE_ACTIVITY_CYCLES_NO_EXECUTE	PMC3	876910168

Metric	Core 0
Runtime (RDTSC) [s]	2.0007
Runtime unhalted [s]	2.6253
Clock [MHz]	3463.7965
CPI	0.5204
Cycles without execution [%]	12.8868
Cycles without execution due to L1D [%]	0.4401
Cycles without execution due to L2 [%]	0.3558
Cycles without execution due to memory [%]	2.5098

TABLE 6.2: L2 and L3 cache for B2 test case.

```
Group 2: L2CACHE
+-----------------------+---------+-------------+
|         Event         | Counter |    Core 0   |
+-----------------------+---------+-------------+
|   INSTR_RETIRED_ANY   |  FIXC0  | 15334903216 |
| CPU_CLK_UNHALTED_CORE |  FIXC1  |  6726320670 |
|  CPU_CLK_UNHALTED_REF |  FIXC2  |  5004954900 |
| L2_TRANS_ALL_REQUESTS |   PMC0  |  1100691015 |
|     L2_RQSTS_MISS     |   PMC1  |   217028917 |
+-----------------------+---------+-------------+

+----------------------+-----------+
|        Metric        |   Core 0  |
+----------------------+-----------+
|  Runtime (RDTSC) [s] |    2.0001 |
| Runtime unhalted [s] |    2.5950 |
|      Clock [MHz]     | 3483.4544 |
|          CPI         |    0.4386 |
|    L2 request rate   |    0.0718 |
|     L2 miss rate     |    0.0142 |
|     L2 miss ratio    |    0.1972 |
+----------------------+-----------+

Group 3: L3CACHE
+--------------------------+---------+------------+
|           Event          | Counter |   Core 0   |
+--------------------------+---------+------------+
|     INSTR_RETIRED_ANY    |  FIXC0  | 3703676320 |
|   CPU_CLK_UNHALTED_CORE  |  FIXC1  | 1405487046 |
|   CPU_CLK_UNHALTED_REF   |  FIXC2  | 1046429388 |
|  MEM_LOAD_RETIRED_L3_HIT |   PMC0  |     534619 |
| MEM_LOAD_RETIRED_L3_MISS |   PMC1  |       3678 |
|     UOPS_RETIRED_ALL     |   PMC2  | 4183712511 |
+--------------------------+---------+------------+

+----------------------+-------------+
|        Metric        |    Core 0   |
+----------------------+-------------+
|  Runtime (RDTSC) [s] |      0.4295 |
| Runtime unhalted [s] |      0.5422 |
|      Clock [MHz]     |   3481.3657 |
|          CPI         |      0.3795 |
|    L3 request rate   |      0.0001 |
|     L3 miss rate     | 8.791235e-07 |
|     L3 miss ratio    |      0.0068 |
+----------------------+-------------+
```

The previous command was run three times to consider some statistics and the average among the measurements were analyzed; the focus was on CPU cycles without execution in total and only due to memory (Fig. 6.1) and L2 and L3 cache misses (Fig. 6.2). Comparing GEOtop 2.0 and 3.0:

- CPU cycles without execution: decrease for all the test cases, more markedly for **Montacini**;
- CPU cycles without execution due to memory: decrease for **B2** and **Montacini** but slightly increase for **snow**: anyway for the latter the increase is of the same order of the measurement variations;
- L2 cache misses: decrease for **snow** and **Montacini** but increase for **B2**, slightly but not within measurement tolerance;
- L3 cache misses: decrease for **B2** but increase for **snow** and **Montacini**, again more than the measurement variations.

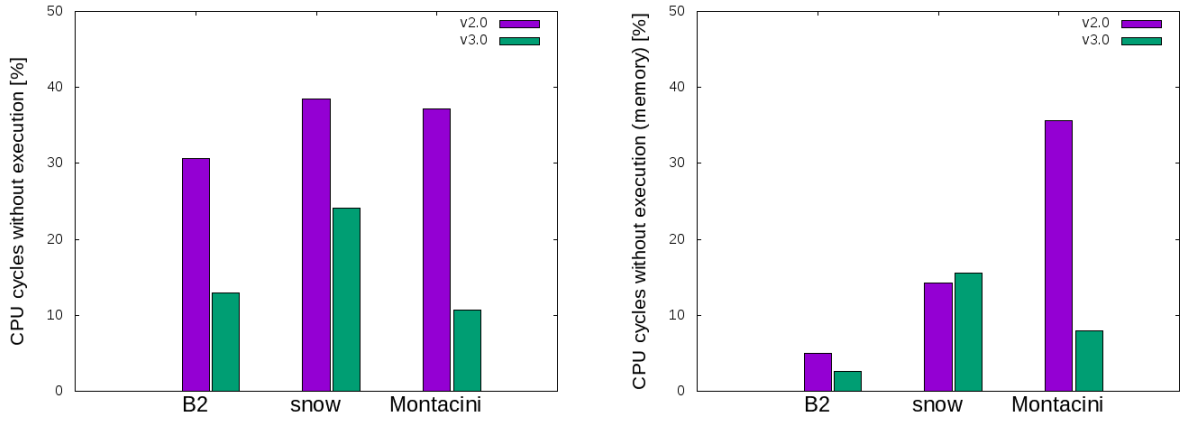


FIGURE 6.1: CPU cycles without execution, total and only due to memory, of the two GEOtop versions for the three test cases.

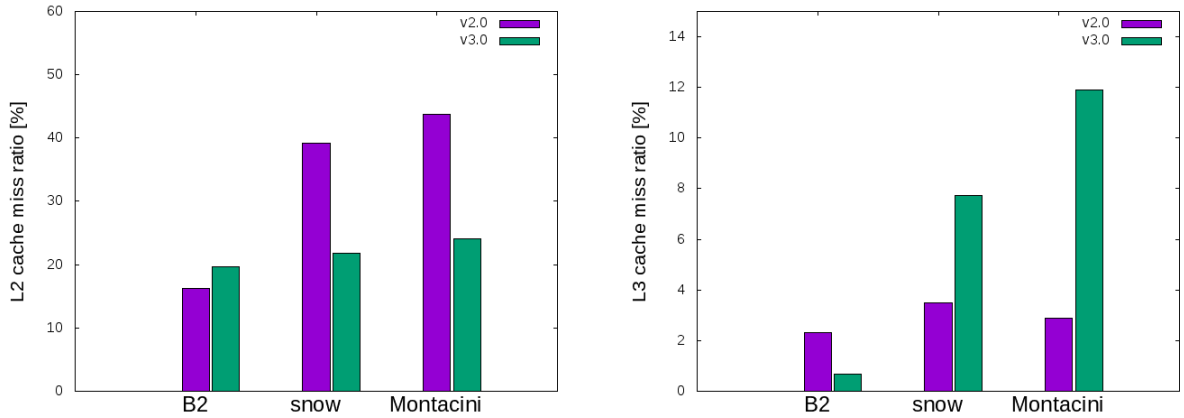


FIGURE 6.2: L2 and L3 cache miss ratios for the two GEOtop versions of the three test cases.

6.2 Callgrind

The command used to produce the output file needed to callgrind was:

```
valgrind --tool=callgrind --dump-instr=yes ./geotop TEST_DIR
```

where

- `-dump-instr=yes` specifies that event counting should be performed at per-instruction granularity and allows for assembly code annotation
- `TEST_DIR` is the directory of the selected **short** test.

The command to read the file using Qcachegrind, included in KCachegrind package, was:

```
./qcachegrind callgrind.out.0123
```

The call graph for the two GEOtop versions for a specific test case displayed the same structure and more or less the same CPU cycles values, indicated for every function inside the rectangular boxes, and the same number of calls, showed over the box.
Analyzing **B2** it can be noticed that for both 2.0 and 3.0 (Fig. 6.3 and 6.4):

- most of the CPU cycles is spent doing calculations (time_loop) and only a few percent in reading the input (get_all_input)
- the water budget calculation dominates (water_balance, and precisely Richards1D since B2 has a 1D setup) but I/O is important (write_output)
- pow() function is too expensive (50%)

Analyzing **snow** it can be noticed that, for GEOtop 2.0 (Fig. 6.5):

- most of the CPU cycles is now spent in reading the input (get_all_input), and especially in computing the sky view factor from the given data;
- energy budget (EnergyBalance) plays anyway an important role (also because water balance is not considered as explained in Chapter 4);
- the function AdaptiveSimpsonAux2 is called millions of time;
- pow() is present (20%).

Instead, for GEOtop 3.0, in which some functions were inlined (see Section 3.5) it can be noticed (Fig. 6.6):

- most of CPU cycle is spent in reading the input;
- all the functions involved in the energy balance and previously written inside util_math.cc were not shown by the profiler, indicating that they involved a low percentage of CPU cycles.

Analyzing **Montacini** it can be noticed that, for GEOtop 2.0 (Fig. 6.7):

- water budget dominates (water_balance, and precisely Richards3D since Montacini has a 3D setup);
- pow() is even more expensive than B2 (70%);

Instead, for GEOtop 3.0 (inline of some functions) it can be noticed (Fig. 6.8):

- water budget dominates but copy_snowvar3D appears in the callgraph;
- pow() is still significant in terms of CPU cycle percentage.

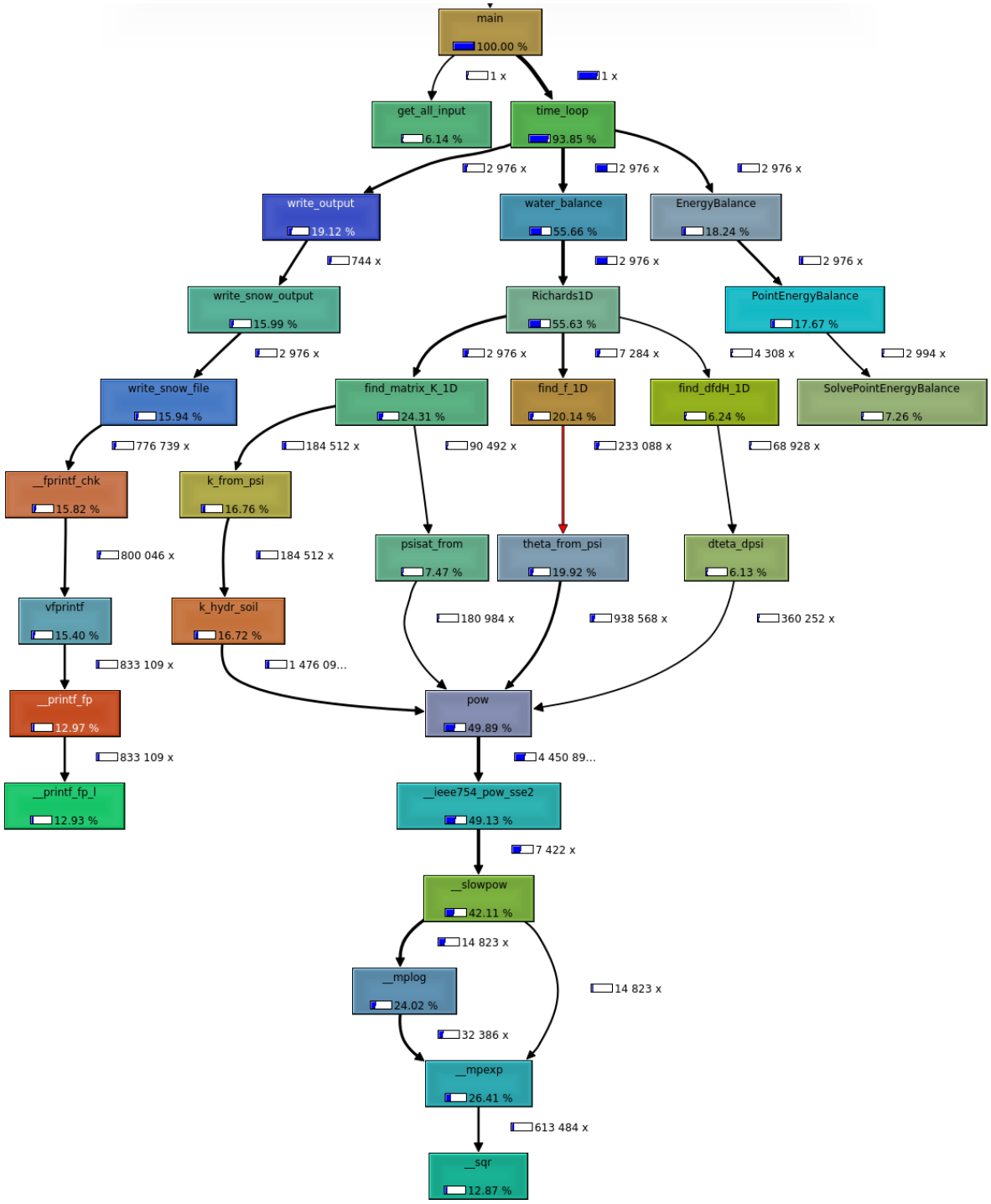


FIGURE 6.3: Callgraph for B2 test case using GEOtop 2.0.

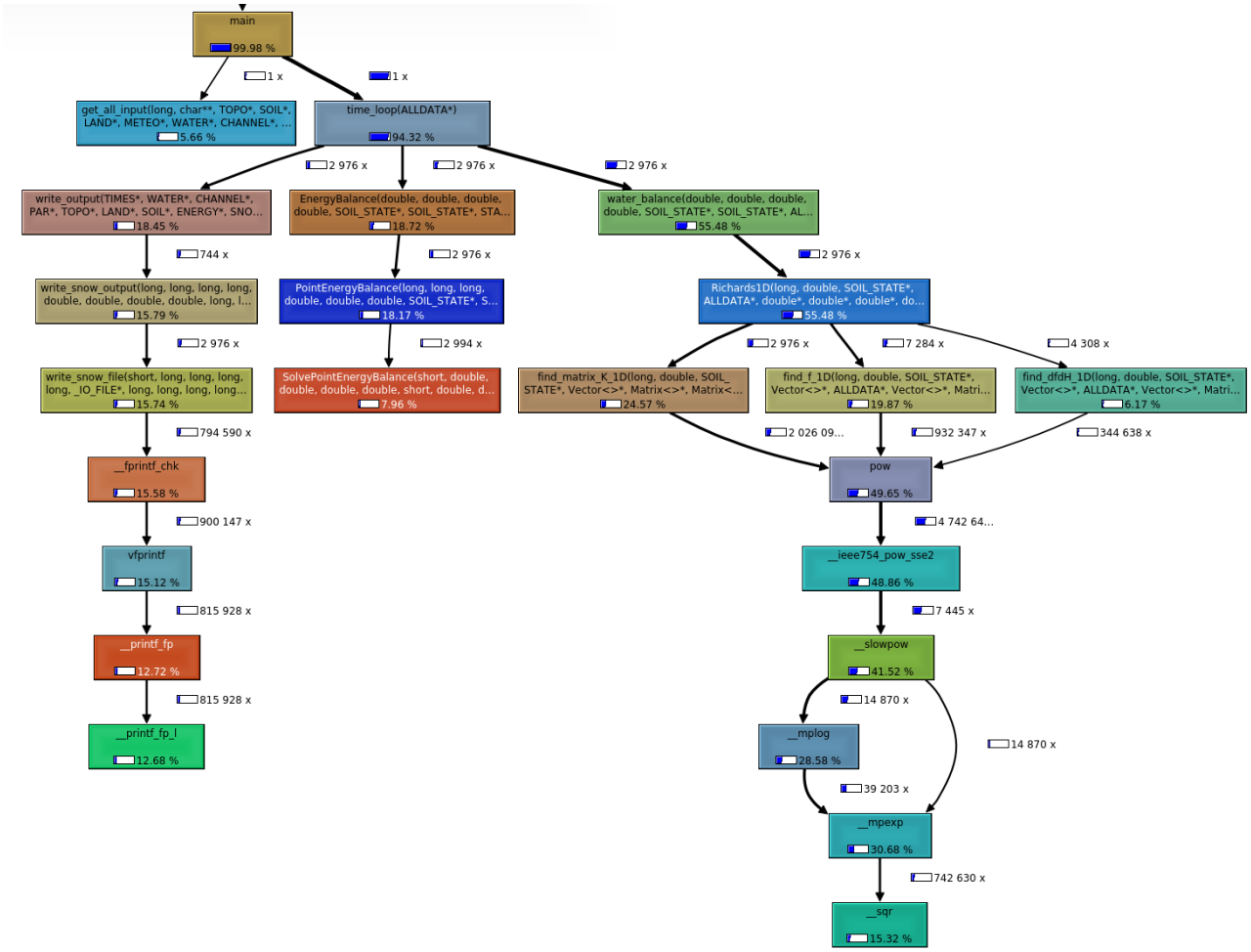


FIGURE 6.4: Callgraph for B2 test case using GEOtop 3.0.

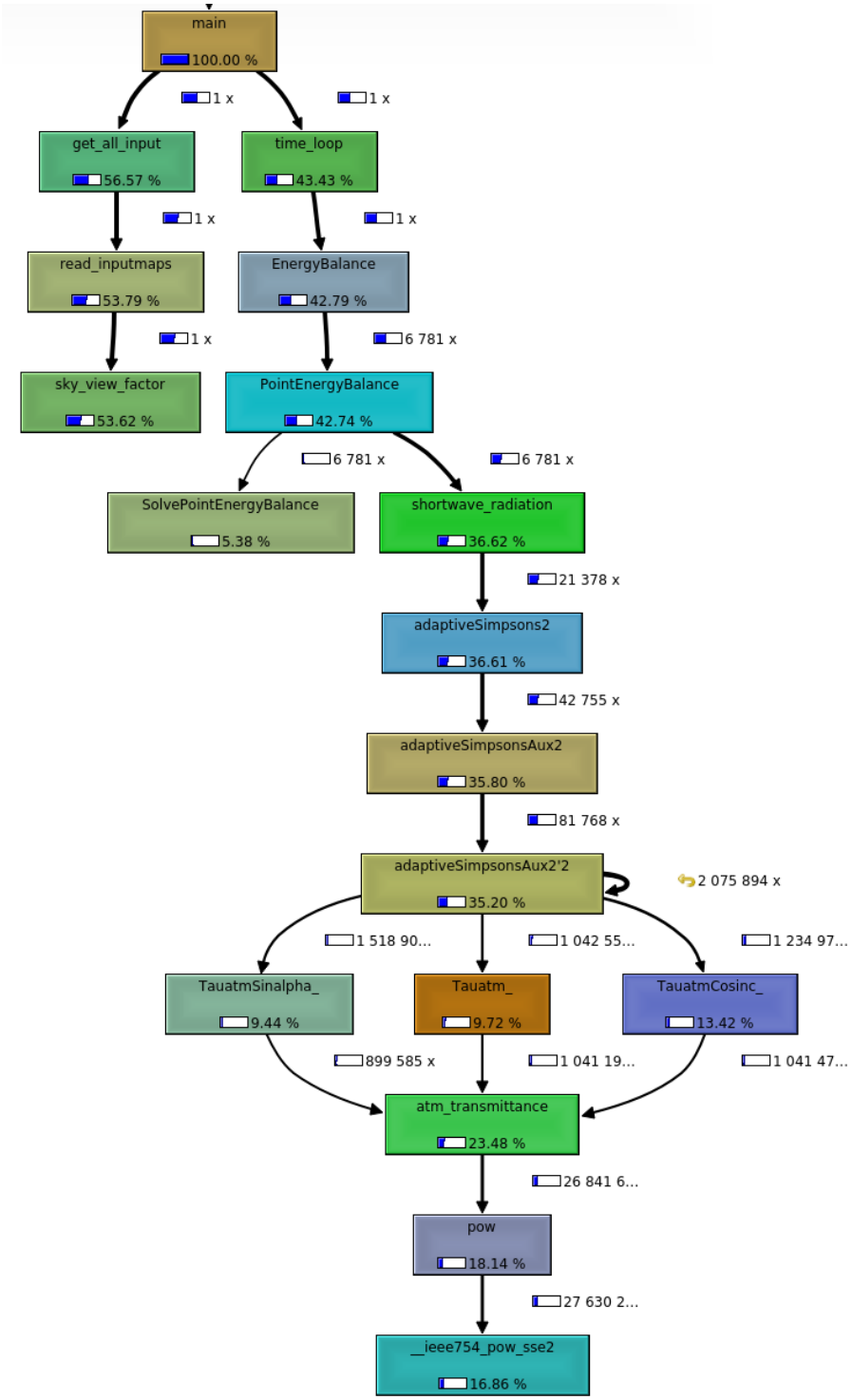


FIGURE 6.5: Callgraph for snow test case using GEOtop 2.0.

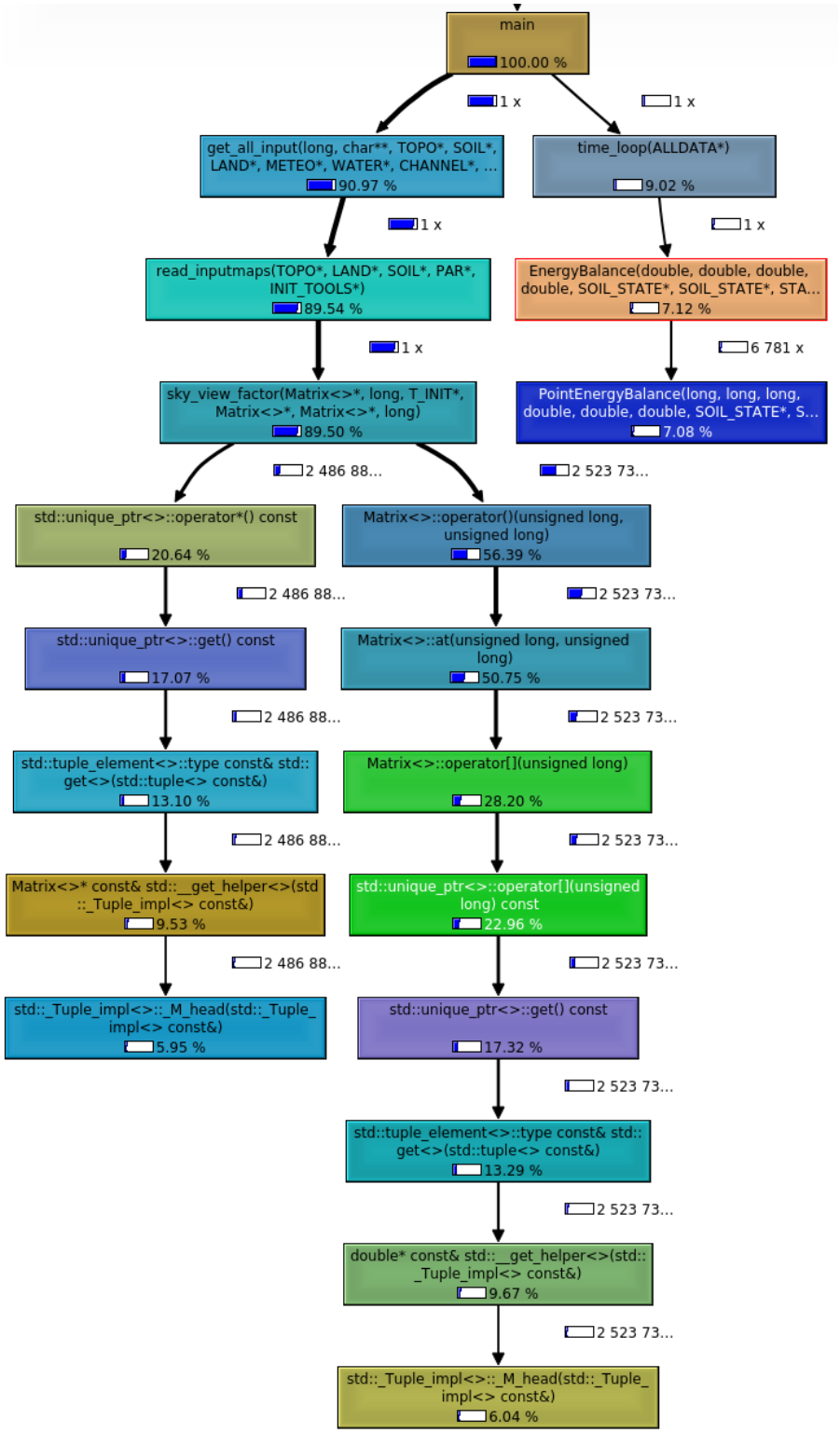


FIGURE 6.6: Callgraph for snow test case using GEOtop 3.0.

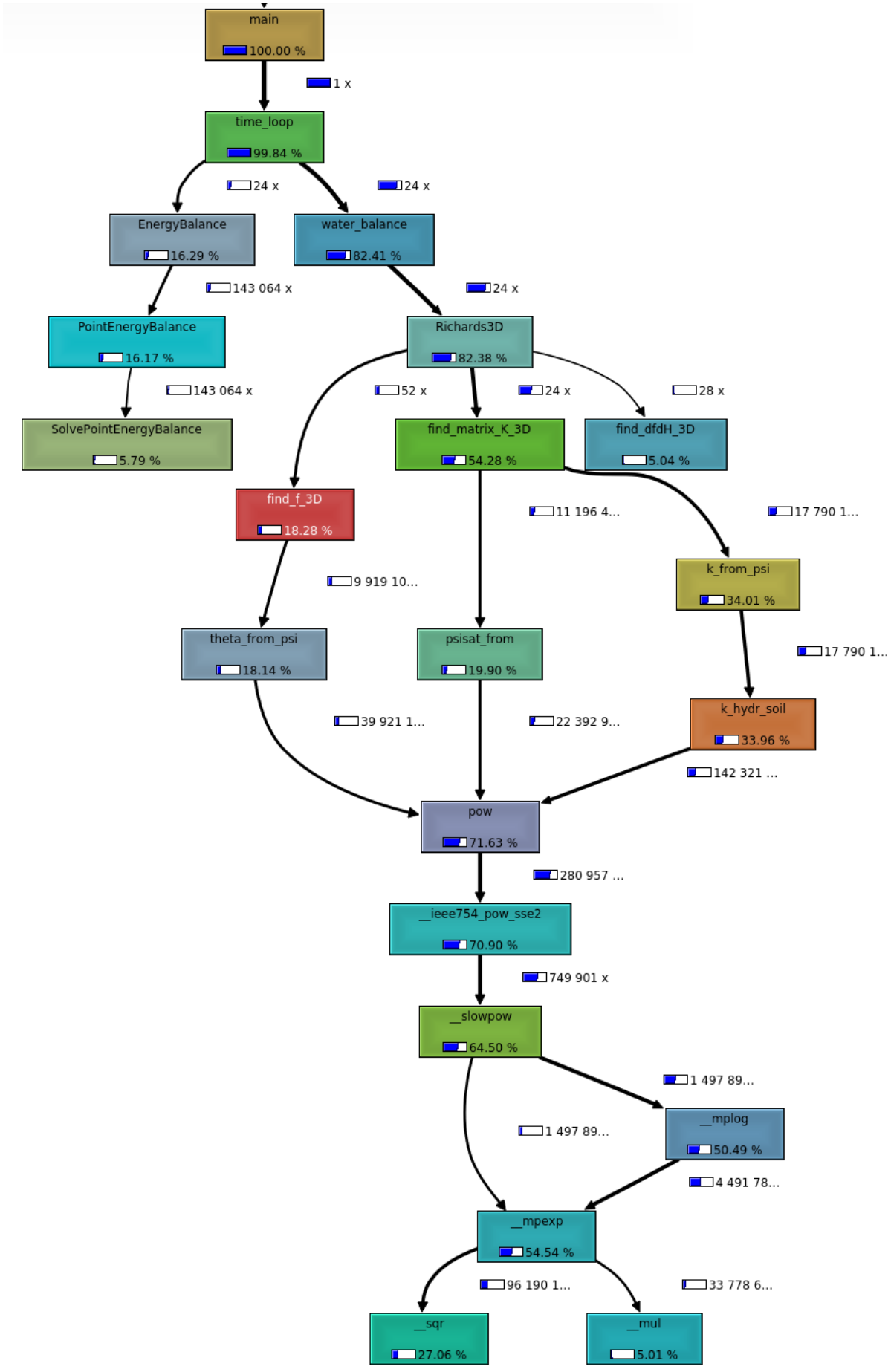


FIGURE 6.7: Callgraph for Montacini test case using GEOtop 2.0.

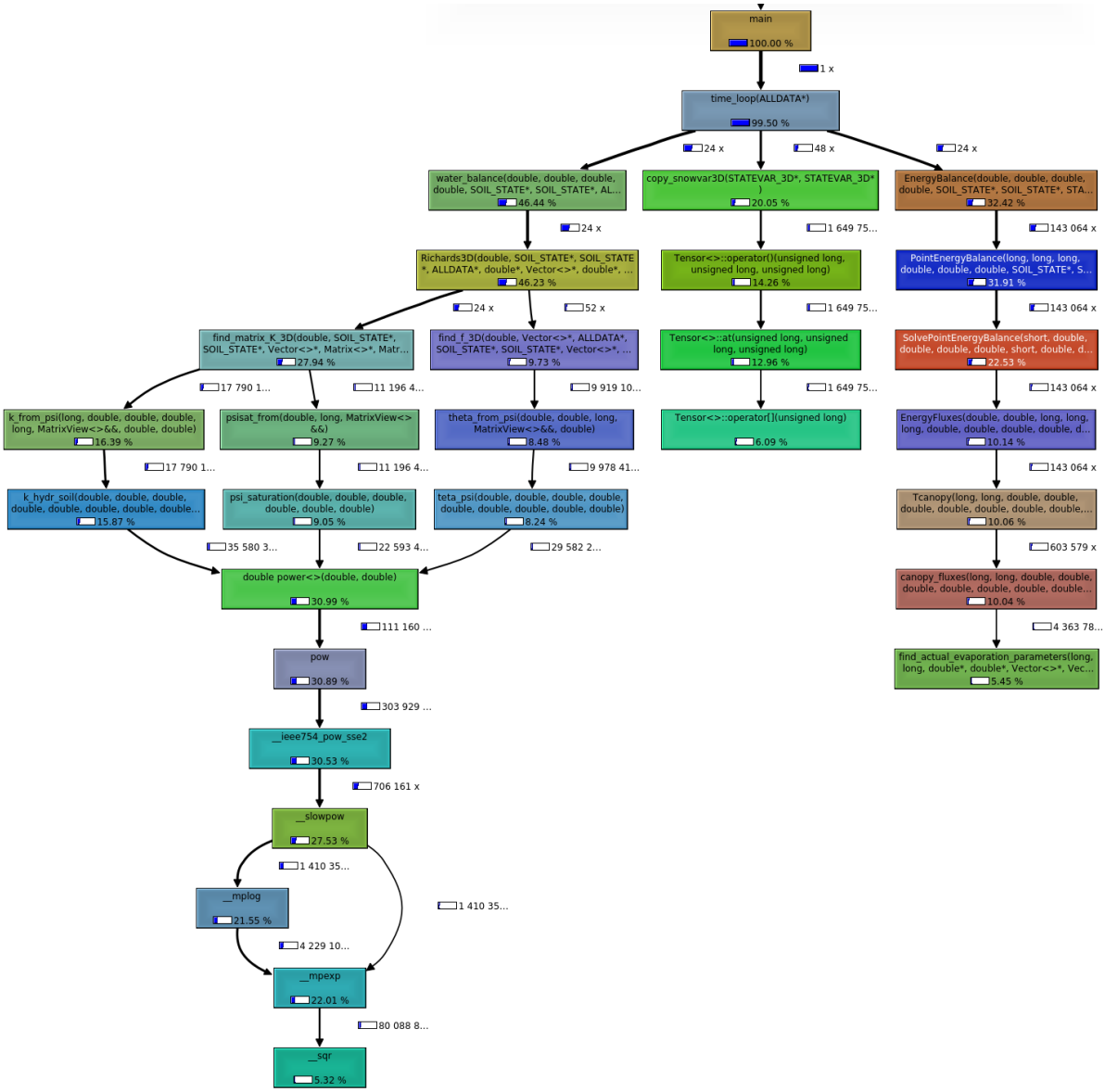


FIGURE 6.8: Callgraph for Montacini test case using GEOtop 3.0.

6.3 Class Timer

A new class Timer was written to measure the number of calls and CPU time, both in absolute values and in percentage compared to the total run, of some relevant functions pointed out by the profiling (and/or already known by GEOtop users and developers to be expensive). In this section the three **short** test cases were analyzed.

```
class Timer {
 void print_summary();
 high_resolution_clock::time_point t_start;
 std::map<std::string, ClockMeasurements> times;
public:
 Timer() : t_start{high_resolution_clock::now()} {}
 ~Timer() {
#  ifndef MUTE_GEOTIMER
     print_summary();
#  endif
 }
 class ScopedTimer;
};
```

LISTING 6.1: Class Timer measuring function calls and CPU times.

For **B2** can be noticed that:

- the results obtained by callgrind regarind water_balance and I/O are confirmed;
- atm_transmittance is the most called function, justifying its inline;
- about 18% of the time is used by input file readings and data structure filling, an action which is done only at the beginning of the simulation;
- about 25% of the time is used by output file readings and data structure filling, an action done at the output time step, which is usually longer than the internal calculation time step;
- about 54% of the time is taken by the two main computational tasks water and energy balance. Those functions are called every computation time steps. This means that, the longer is the simulation period, the bigger is the cumulative time required by those functions.

For **snow** it can be noticed that:

- even if Callgrind indicated that most of the CPU cycles were spent in reading the input, the most expensive part on a time basis is the energy balance, summing up to the 57% of the simulation time;
- atm_transmittance is again the most called function, and this time it has more influence in the total CPU time than for B2;
- sky_view_factor is quite time-expensive, even if not as much as in terms of CPU cycles. However, this function is called only once at the beginning of the simulation.

TABLE 6.3: Output of the class Timer for B2 test case.

Total CPU time elapsed since start		1.98s	
Section	no. calls	CPU time	% of total
atm_transmittance	44730	0.04s	1.9%
copy_snowvar3D	5952	0.02s	1.0%
SolvePointEnergyBalance	2994	0.19s	9.6%
Richards1D	2976	0.68s	34.4%
water_balance	2976	0.68s	34.6%
write_output	2976	0.49s	24.7%
EnergyBalance	2976	0.39s	19.6%
PointEnergyBalance	2976	0.37s	18.4%
find_matrix_K_1D	2976	0.30s	15.2%
write_snow_file	2980	0.29s	14.9%
get_all_input	1	0.37s	18.5%

Actually CPU cycles can have difference time duration, so there is no guarantee that the relative importance of a function in terms of CPU cycles is the same of its relevance on CPU times basis.

TABLE 6.4: Output of the class Timer for snow test case.

Total CPU time elapsed since start		9.04s	
Section	no. calls	CPU time	% of total
atm_transmittance	3102463	2.90s	32.1%
PointEnergyBalance	6781	5.18s	57.3%
SolvePointEnergyBalance	6781	0.54s	6.0%
write_snow_file	24	0.00s	0.00%
copy_snowvar3D	4	0.03s	0.3%
EnergyBalance	1	5.18s	57.3%
get_all_input	1	3.77s	41.7%
sky_view_factor	1	3.35s	37.1%
write_output	1	0.01s	0.00%

For **Montacini** it can be noticed that:

- water_balance is the most time-consuming function, having higher relevance in terms of CPU-time than CPU-cycles;
- copy_snowvar3D appears but with less importance in time than cycles;
- atm_transmittance is again the most called function but it does not affect too much the total CPU time.
- Input/Output functions require a relatively small amount of time.

TABLE 6.5: Output of the class Timer for Montacini test case.

Total CPU time elapsed since start		86.5s	
Section	no. calls	CPU time	% of total
atm_transmittance	3135930	2.32s	2.7%
PointEnergyBalance	143064	21.00s	24.3%
SolvePointEnergyBalance	143064	10.16s	11.7%
find_f_3D	52	13.96s	16.2%
copy_snowvar3D	48	0.99s	1.1%
Richards3D	24	63.18s	73.1%
water_balance	24	63.24s	73.1%
find_matrix_K_3D	24	40.98s	47.4%
EnergyBalance	24	21.17s	24.5%
write_output	24	0.51s	0.6%
get_all_input	1	0.45s	0.5%

The output measurements for **long** tests are instead discussed in Chapter 8, since it will be used to compare GEOtop 3.0 with and without optimizations, activated by flags.
The measurements are active by default; if someone is not interested in, he/she has to type in the build directory

```
meson configure -DMUTE_GEOTIMER=true
```

and nothing regarding time will be printed.

Chapter 7

Optimizations

In order to make GEOtop 3.0 more efficient than the 2.0, some optimizations were added:

- a fast **maths optimization**, enabled by typing in the build directory `meson configure -DMATH_OPTIM=true`;
- the possibility to use **OpenMP**, activated by typing `meson configure -DWITH_OMP=true`;
- an automatic **vectorization**, used after typing in the same folder `-Dcpp_args="-march=native"`.

7.1 Maths optimization

The profiling showed that the pow() is very much used by the analyzed test cases. Since it is an expensive function (https://streamhpc.com/blog/2012-07-16/how-expensive-is-an-operation-on-a-cpu/) it could replaced by other less CPU-intense operations.
Precisely when the exponent of the power was 2, the

```
std::pow(a,2)
```

was replaced by

```
pow_2(a) ((a)*(a))
```

defined as a macro inside the new header file named math.optim.h; in this way the run time should decrease since a less-expensive operation, a simple multiplication, is involved and the function is computed at compile time.
This substitution involved all the source files computing that operation; they were inside the folders:

- libraries: util_math.h and geomorphology.0875.cc;
- geotop: water.balance.h, meteo.cc, meteodistr.cc, output.cc, PBSM.cc, pedo.funct.h, vegetation.cc blowingsnow.cc, input.cc, radiation.cc, turbulence.cc, snow.cc

Then, after applying the following property of logarithms:

$$a^b = e^{b \cdot log(a)} \tag{7.1}$$

another new power function was defined (see code box 7.1).

```
template <class T>
T power(const T a, const T b){
#ifdef MATH_OPTIM
    return std::exp(b*std::log(a));
#else
    return std::pow(a,b);
#endif
}
```

LISTING 7.1: User defined power function.

The function exp() is less CPU-expensive than pow() (https://streamhpc.com/blog/2012-07-16/how-expensive-is-an-operation-on-a-cpu/) so its usage should decrease the run time. Anyway the new expression is valid only for positive a, since the argument of the logarithm must be > 0, so the new power could not be used in all the code parts.
The new function replaced the pow() in radiation.cc for the function atm_transmittance and in pedo.func.h for the evaluation of the parameter psisat, that is the saturated pressure head written using Genuchten, 1980 formula.

$$\psi_{sat} = \frac{1}{\alpha}\left[\left(\frac{\theta - \theta_r}{\theta_s - \theta_r}\right)^{1/m} - 1\right]^{1/n} \tag{7.2}$$

Actually the parameters involved in atm_transmittance as the bases of power functions are angles, masses and thicknesses and they always positive for physical reasons (Iqbal, 1983; Gueymard, 1989). Also regarding the evaluation of psisat, the physical parameter involved as basis are soil water content and they are always in range between 0 and 1 (https://en.wikipedia.org/wiki/Water_content).
To be sure to avoid negative bases, an empirical check was done collecting all the values of both bases and exponents of the previously mentioned functions. Changes were implemented only for the power operations in functions with no negative bases.

7.2 OpenMP parallelization

The GEOtop model, as many codes written more than ten years ago, has been written with a serial approach, which is now obsolete for modern multi-threads computer architectures. The model works on a raster basis, with many nested functions (i.e. Fig. 2.4). A full code parallelization with MPI, even if it could have allowed to exploit multiple cores on modern architectures, would have needed a complete code rewriting, and by the way, it was outside the scope of the thesis. Here the work has been limited to improve the efficiency of computationally expensive parts of the code, but already thread safe.
A parallelization with OpenMP was chosen because most of the functions use the same data, so a shared memory paradigm seemed to be suited for the case; besides, it can be added incrementally, without huge efforts.

Therefore, some functions, relevant in terms of CPU cycles and times and especially thread-safe, were parallelized with OpenMP. Precisely the profiling suggested to work on:

- get_all_input, in input.cc (~67% of CPU time for B2 and 41% for snow)
- sky_view_factor, in geomorphology.0875.cc (~37% of CPU time for snow)
- find_f_3D, in water.balance.h (~16% of CPU time for Montacini)
- copy_snowvar3D, in snow.cc (~1% of CPU time for Montacini)

The functions do the following:

get_all_input: reads multiple input file for meteorological stations.
This function could be time expensive, but is called only once before the main time loop of the code. The implemented parallelization allows to read and allocate the data of more input file on the same time. It is expected significant performance improvement if test cases with many input files (i.e. many meteo stations), but short computation time. A typical application is the operational MySnowMaps tool (Dall'Amico, Endrizzi, and Tasin, 2018), where the model is run every day, but over a large domain (the whole Alps),

sky_view_factor: calculates the sky view factor, the fraction of visible sky from an observation point (https://it.wikipedia.org/wiki/Sky-view_factor), for each pixel.
This function could be time expensive, but is called only once before the main time loop of the code. The implemented parallelization allows to elaborate on the same time the loop on the DEM matrix. It is expected significant performance improvement if test cases large domains, but short computation time.

find_f_3D: calculates an array containing drainage terms at the bottom and at the border.
The parallelization speeds up the calculation for large domains and many soil layers. This function is called inside the function water_balance every calculation time step. It is expected that the optimization could be effective for long time simulations.

copy_snowvar3D: creates matrices of variables that will be copied in the output files (by other functions).
This function is called every output time step, which can be configured by the user (usually between one hour and one day). It is activated only it the used want in output detailed information on snow properties. It is expected an effective optimization only for operational conditions when snow processes are considered and a frequent output time step is needed.

Some for loops were parallelized and the involved variables were declared to be: (i) shared, if they had to be shared among threads; (ii) private if they had to be private to each thread and (iii) firsprivate if they had to be private to each thread but they were initialized outside the for loop. (https://www.openmp.org/).

7.3 Automatic vectorization

The automatic vectorization is a special case of automatic parallelization, where a computer program is converted from a *scalar* implementation, which processes a single pair of operands at a time, to a *vector* implementation, which processes one operation on multiple pairs of operands at once (https://en.wikipedia.org/wiki/Automatic_vectorization).
For example the following code lines:

```
for (i = 0; i < 1024; i++)
    C[i] = A[i]*B[i];
```

could be vectorized to look something like:

```
for (i = 0; i < 1024; i+=4)
     C[i:i+3] = A[i:i+3]*B[i:i+3];
```

The added compiling flags to let the compiler transforms loops to vector operations were:

- `-march=native`, applied both to Intel Core and Intel Xeon, enabling all instruction subsets supported by the local machine;
- `-march=ivybridge`, applied only when using Intel Xeon,enabling all the instruction subsets supported by am Intel Ivy Bridge CPU with 64-bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AVX, AES, PCLMUL, FSGSBASE, RDRND and F16C instruction set support (https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html).

7.4 Combination

Theoretically, the best configuration, meaning the one having the lowest run time and the same output results of GEOtop 2.0 for all the three test cases, should be be obtained combining several optimization flags.

Chapter 8

Optimization results

In this section, changes in time and eventually in output results of the optimized version of GEOtop 3.0 are compared to GEOtop 2.0. Only the **long** tests were analyzed, since the short tests lasted for a too short time (few seconds for 1D and few minutes for 3D) to be able to appreciate significant differences in the output files.

8.1 Default 3.0

Run times for GEOtop 2.0 and default 3.0 for Intel Core and Intel Xeon are compared in Fig. 8.1 for the test cases:

- **B2**: the new version is a bit slower than the 2.0, even if the difference is comparable to the standard deviation of measurements;
- **snow**: the 3.0 is a bit faster, even if the difference is again comparable to the standard deviation of the measurements;
- **Montacini**: there is a valuable time decrease in using the 3.0.

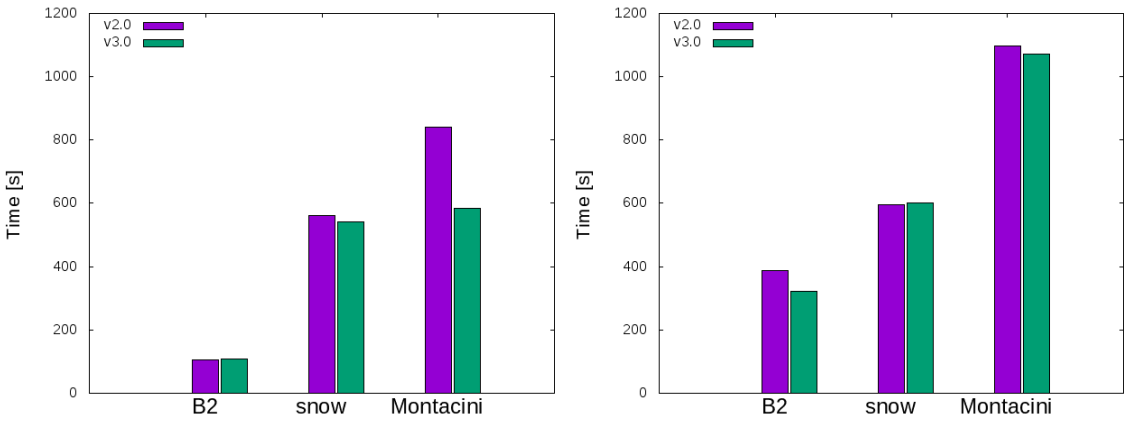


FIGURE 8.1: Run time for GEOtop 2.0 and default 3.0 for Intel Core and Intel Xeon.

8.2 Maths optimization

Activating the flag MATH_OPTIM, the run time decreased significantly for all the test cases, especially for **Montacini**, for which the time almost halved, as shown in Fig. 8.2.
Unfortunately this optimization changed the output results for **B2** and **snow**, so the test_runner, comparing the output values of 2.0 and 3.0 using absolute and relative tolerances of 10^{-5}, failed. To understand if this tests failure is above or below tha expected models accuracy for the different output variables, a scientific validation of the failing test cases is performed in Chapter 9.

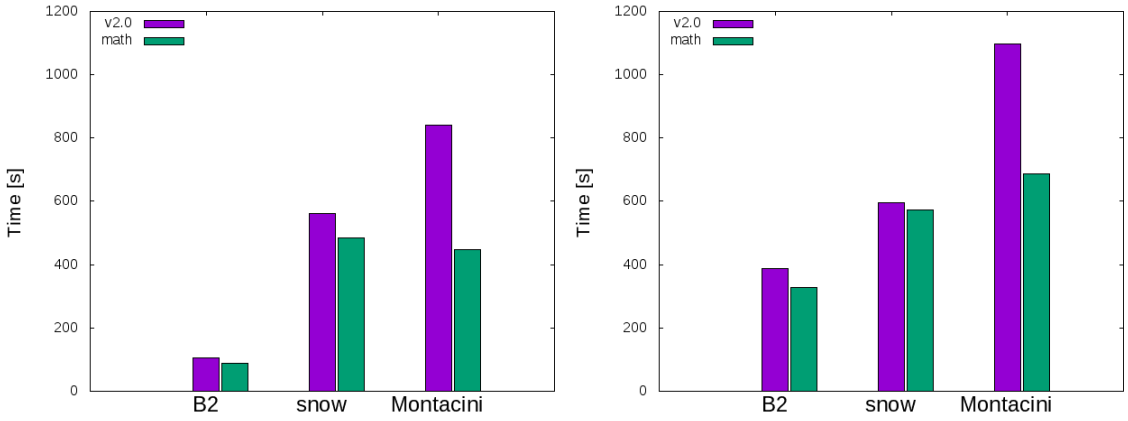


FIGURE 8.2: Run time for GEOtop 2.0 and 3.0 with math optimization for Intel Core and Intel Xeon.

8.3 OpenMP parallelization

The flag WITH_OMP was activated and the results were analyzed for a number of threads equal to 1, 2, 4, 8 and 16 on both Intel Core and Intel Xeon. The results in terms of CPU time for the total run were different among the tests (Figures 8.3 and 8.3).

For **B2** the time increased together with the number of threads, showing an inverse scalability. In this test case, only two of the four parallelized functions are used and precisely get_all_input and copy_snowvar3D. Since **B2** is a 1D test with only one file containing meteorological data and the variables involved in the coping part are few, the time increase is due to the overhead of creating multiple threads whose work will be quite limited.

For **Montacini** the time decreased using more threads but very slowly, gaining only a few percentage. In this test case, three of the four parallelized functions are used and precisely get_all_input, copy_snowvar3D and find_f_3D. The time decreases because, even if for the first two functions their relative importance on time basis is lower for long test (compared to the short), find_f_3D is quite relevant. Actually it is inside the time loop, so it works at each iteration and also for every active cell. Since this 3D test case has 16'848 pixels (and a part of them will be active), the improvement is noticeable. Anyway, there are other important functions that are not parallelized, hence the not so high time decrease.

For **snow** nothing changed. In this test case, three of the four parallelized functions are used and precisely get_all_input, sky_view_factor and copy_snowvar3D. Even if there are seven meteorological stations so seven input files that could be handled separately by threads (get_all_input) and the matrix for the sky view factor calculation has thousands of cells (10'140 total pixel), these two functions are anyway outside the for loop, which is usually the main part for a 3D simulation. Actually the relative importance of these functions on long tests is very limited; hence the absence of noticeable changes.

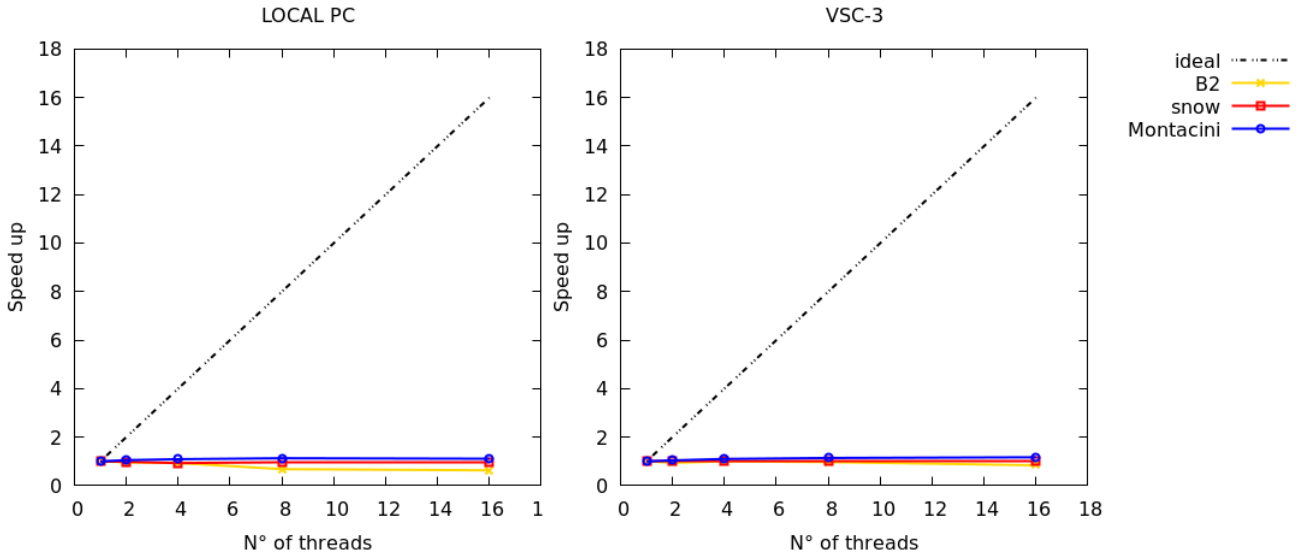


FIGURE 8.3: Speed up using OpenMP on Intel Core and Intel Xeon: whole picture.

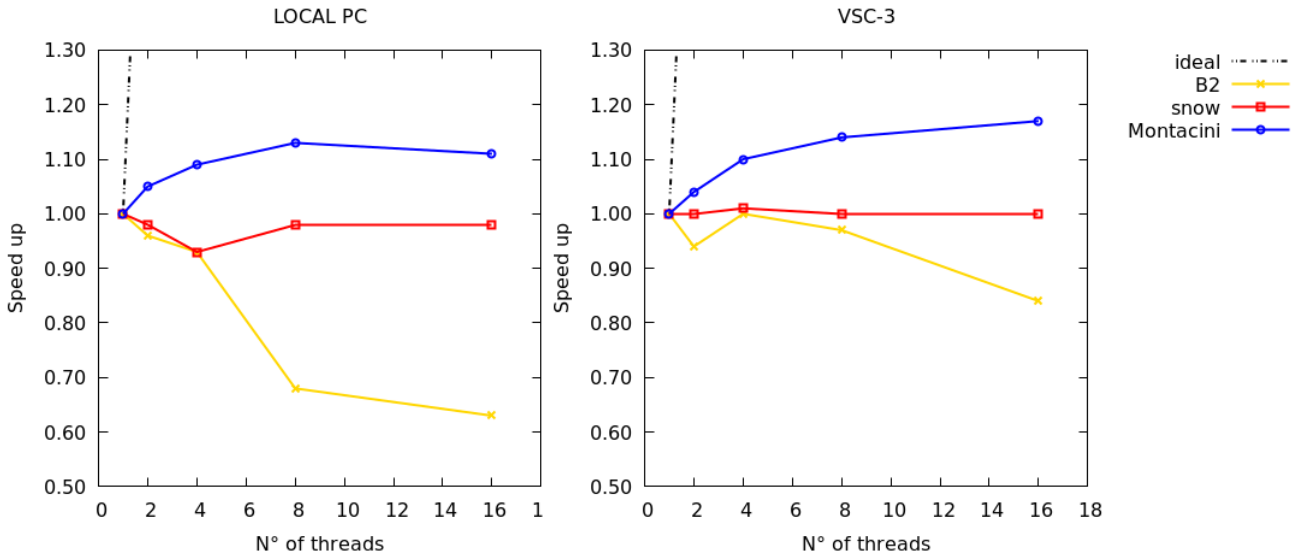


FIGURE 8.4: Speed up using OpenMP on Intel Core and Intel Xeon: zoom in.

Analyzing the scalability of the parallelized functions individually it can be noticed that only find_f_3D shows some scalability, even if sublinear; the other three do not scale (Fig. 8.5). in fact find_f_3D is the only parallelized function that is used for every computation time step. Actually, their CPU times is quite small (< 5 s) so the threads do just a few work.

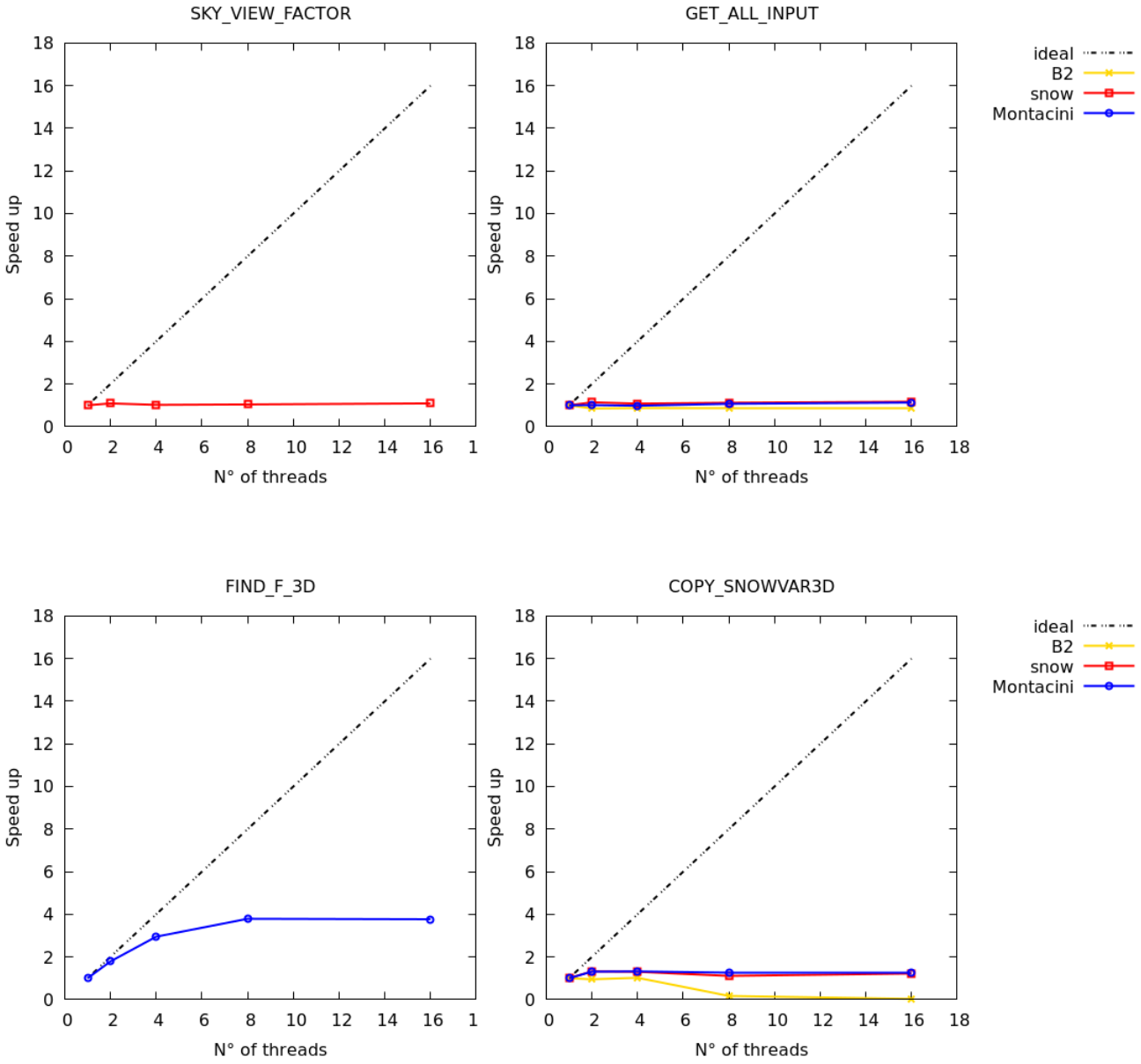


FIGURE 8.5: Speed up of the parallelized functions on my pc.

Decreasing the speed-up range to zoom in (Fig. 8.6), it is visible that:

- copy_snowvar3D: for snow and Montacini there is a slight increase up to 4 threads but then nothing changes basically, while B2 shows again an inverse scalability; this happens because the functions works with matrices but B2 is a 1D test case;
- sky_view_factor: nothing changes, also because the parallel section does not cover all the function but it is just a part;
- get_all_input: B2 shows again an inverse scalability since it has only 1 meteorological stations as input; for Montacini, having 4 input stations, nothing changes while for snow, having 7 input stations, there is a slight increase for two threads but then nothing changes.

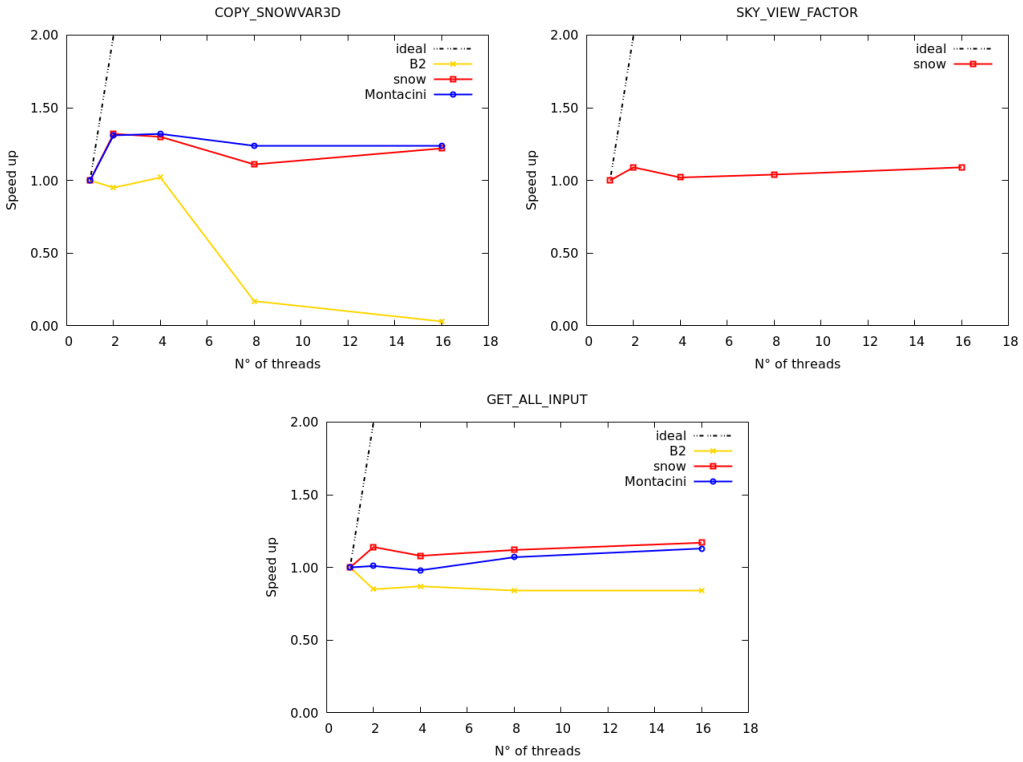


FIGURE 8.6: Speed up of the parallelized functions: zoom.

So, the best test case for each parallelized function could have been, for:

- get_all_input: lots of meteorological stations (i.e., climatic model) and/or short test cases with a few computations;
- sky_view_factor: big matrices and short test cases;
- copy_snowvar3D: big matrices and short output time frequency or long simulation time; actually this function works at every output time step (for example one month means that this function is executed 30 times);
- find_f_3D: a lot of time spent inside the time loop, so a very CPU-intensive case.

8.4 Vectorization

Using `-march=native` on Intel Core the time increased for **B2** but decreased for both the 3D tests, especially **Montacini**. Unfortunately, the output results differed compared to GEOtop 2.0 for B2 and snow.
Using on `-march=native` or `-march=ivybridge` on VSC-3 led to the same results: for **B2** there was more or less no difference compared to 3.0 without the vectorization flag but for **snow** and **Montacini** there was a time decrease, even if smaller than the one on Intel Core. Probably the activated vectorization was not so "aggressive"; in this case the output values were equal to GEOtop 2.0, avoiding the need of a scientific validation (Fig. 8.7).

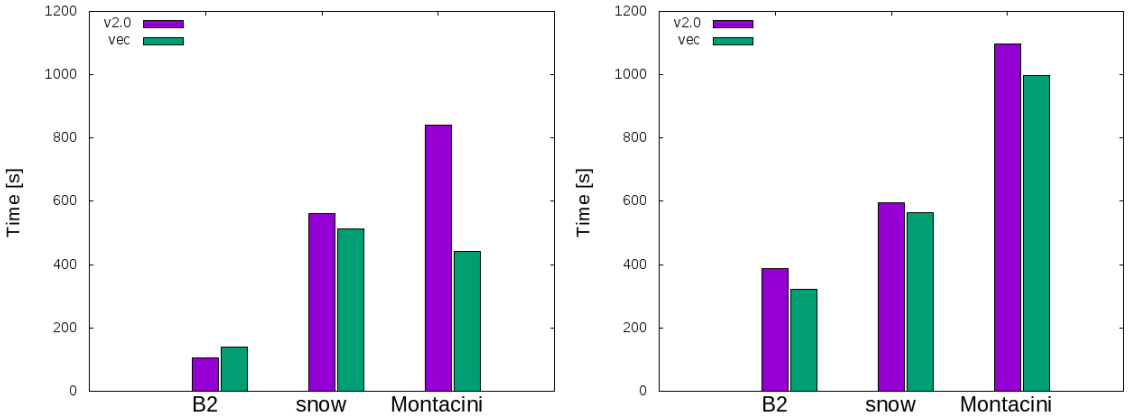


FIGURE 8.7: Run time for GEOtop 2.0 and 3.0 with vectorization for Intel Core and Intel Xeon.

8.5 Combination

The best configuration was obtained for:

- Intel Core: using OpenMP (1 thread for the 1D and 8 threads for 3D);
- Intel Xeon: using OpenMP with 16 threads and `-march=ivybridge`.

As shown in Fig. 8.8, this leads to a marginal performance changes for the **B2** and **snow** test cases, but to an improvement of almost 50% for the **Montacini** 3D test, which is the most computationally expensive one.

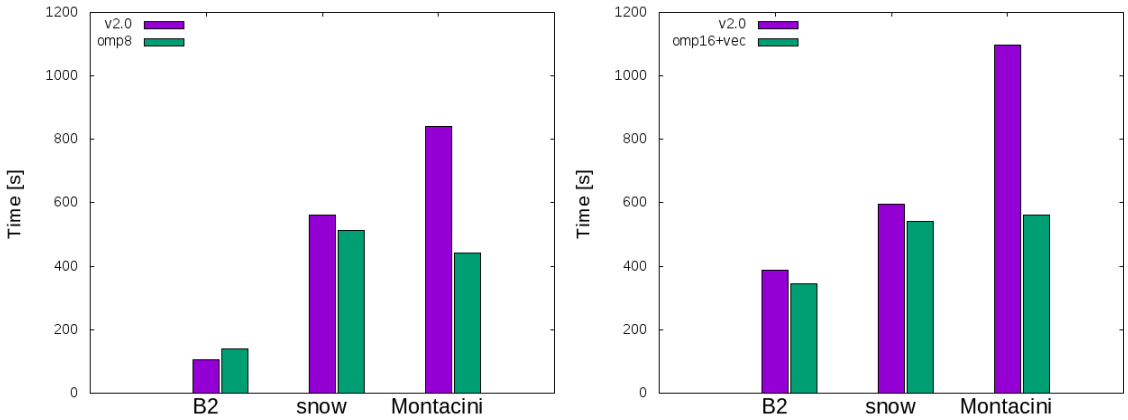


FIGURE 8.8: Run time for GEOtop 2.0 and 3.0 in the best case (shortest time for 3.0) for Intel Core and Intel Xeon.

Chapter 9

Scientific validation

This section provides a scientific validation of the failing test cases to understand if the tests failure was above or below the expected models accuracy for the different output variables. In Tab. 9.1 it is reported for each test case when there are differences in output values between GEOtop 2.0 and 3.0 above the chosen threshold of 10·5, after applying a specific optimization. For example, using **maths optimization**, B2 (1D setup) and snow (3D setup), the tests fail.

TABLE 9.1: Results of outputs comparison between GEOtop v2.0 and v3.0. A test fails when there is a difference of more than 10^{-5} in the output files.

Test case	Maths optimization	OpenMP parallelization	Automatic vectorization
B2	FAIL	OK	FAIL
snow	FAIL	OK	FAIL
Montacini	OK	OK	OK

To understand if the results could be still considered scientifically valid, specific tolerances for each output variable have been defined (see Tab.9.2). Those values have been based on the accuracy in observing or measuring the specific variable. For an explanation of the physical meaning of the output variables please refer to the GEOtop manual (Endrizzi et al., 2011).
Output time series and output maps have been further analyzed when differences exceeded this tolerance.

TABLE 9.2: Tolerance units for each output variable.

Process	Precipitation	ET	Mass_balance_error	Snow/Water con canopy	Temperature	Energy fluxes	Wind_speed	Wind_direction	Relative_Humidity
Units	mm	mm	mm	mm	C	W/m2	m/s	deg	-
Tolerance	0.1	0.1	0.1	0.1	0.1	1	0.1	1	0.01
Process	Canopy_fraction	LSAI	Pressure	z0veg	d0veg	Qveg/sur/air	Lobukhov	Decay_of_K_in_canopy	snow_depth[mm]
Units	-	[m2/m2]	mbar	[m]	[m]	-	m	-	mm
Tolerance	0.01	0.1	1	0.01	0.01	0.0001	0.1	0.1	10
Process	snow_water_equivalent[mm]	snow_density[kg/m3]	snow_temperature[C]	snow_melted[mm]	snow_subl[mm]	snow_blown_away[mm]	snow_subl_while_blown[mm]	glac_depth[mm]	glac_water_equivalent[mm]
Units	mm	Kg/m3	C	mm	mm	mm	mm	mm	mm
Tolerance	1	10	0.1	0.1	0.1	0.1	0.1	100	1
Process	glac_density[kg/m3]	glac_temperature[C]	glac_melted[mm]	glac_subl[mm]	lowest_thawed_soil_depth[mm]	highest_thawed_soil_depth[mm]	lowest_water_table_depth[mm]	highest_water_table_depth[mm]	snowDav
Units									days
Tolerance	10	0.1	1	1	10	10	10	10	1
Process	SWEav	Tav	Tsav	perc.SFA	perc.SCA				
Units	mm	C	C	%	%				
Tolerance	1	0.1	0.1	1	1				
File	pslzNNNN.TXT	snowDepthNNN.txt	snowIceNNNN.txt	snowLiqNNNN.txt	snowTNNNN.txt	snodepthN*.asc	snowdurationN00*.asc	snowmeltedN000*.asc	snowsublN00*.asc
Units	mm	mm	mm	mm	mm	mm	mm	mm	mm
Tolerance	10	10	1	1	0.1	1	0.1	1	1

9.1 B2 (1D test case)

The failing output files, found in output-tabs folder, were basin.txt, point0001.txt, psiz0001.txt, snowDepth0001.txt, snowIce0001.txt, snowLiq0001.txt, snowT0001.txt (*001.txt since the test case involves only one point, the meteorological station B2).

9.1.1 basin.txt

The significant differences are resumed in Tab.9.3: for every variable, are reported the variable name (*Variable*), the maximum (*Max*) and mean absolute difference value, both considering only the different cases (*Mean*) and all the output (*Mean_tot*), the number *N* of time steps exceeding the specific threshold, the total number of time steps (*Tot*) and the number of time steps exceeding the specific threshold expressed as percentage (*N%*).
The difference are locally meaningful, but they regard a very limited number of time steps (< 0.02%) of the output time series. For this reason, the results could be considered scientifically valid. A more detailed analysis is performed for some variables.

TABLE 9.3: Output variables (file basin.txt) whose values are different between v2.0 and v3.0 for B2 test case.

Variable	Max	Mean	Mean_tot	N	Tot	N %
Tsurface [°C]	0.747	0.467	2.605E-05	7	54767	0.013
Tvegetation [°C]	0.388	0.230	2.017E-05	6	54767	0.011
H [W/m2]	5.113	5.113	1.625E-05	1	54767	0.002
LW [W/m2]	2.358	1.348	-3.111E-05	1	54767	0.002
Hv [W/m2]	3.158	1.834	6.462E-05	3	54767	0.005
Swv [W/m2]	2.534	2.534	4.691E-05	1	54767	0.002
Lwv [W/m2]	1.890	1.518	-8.561E-05	2	54767	0.004
Swin [W/m2]	2.897	2.897	5.290E-05	1	54767	0.002
Lwin [W/m2]	8.613	8.613	1.573E-04	1	54767	0.002

A more detailed analysis has been performed for some variables, for example for $Tsurface$, which is the modelled soil surface temperature. As it can be seen in Fig. 9.1, the difference among the models is limited, and below the typical variability of this variable.

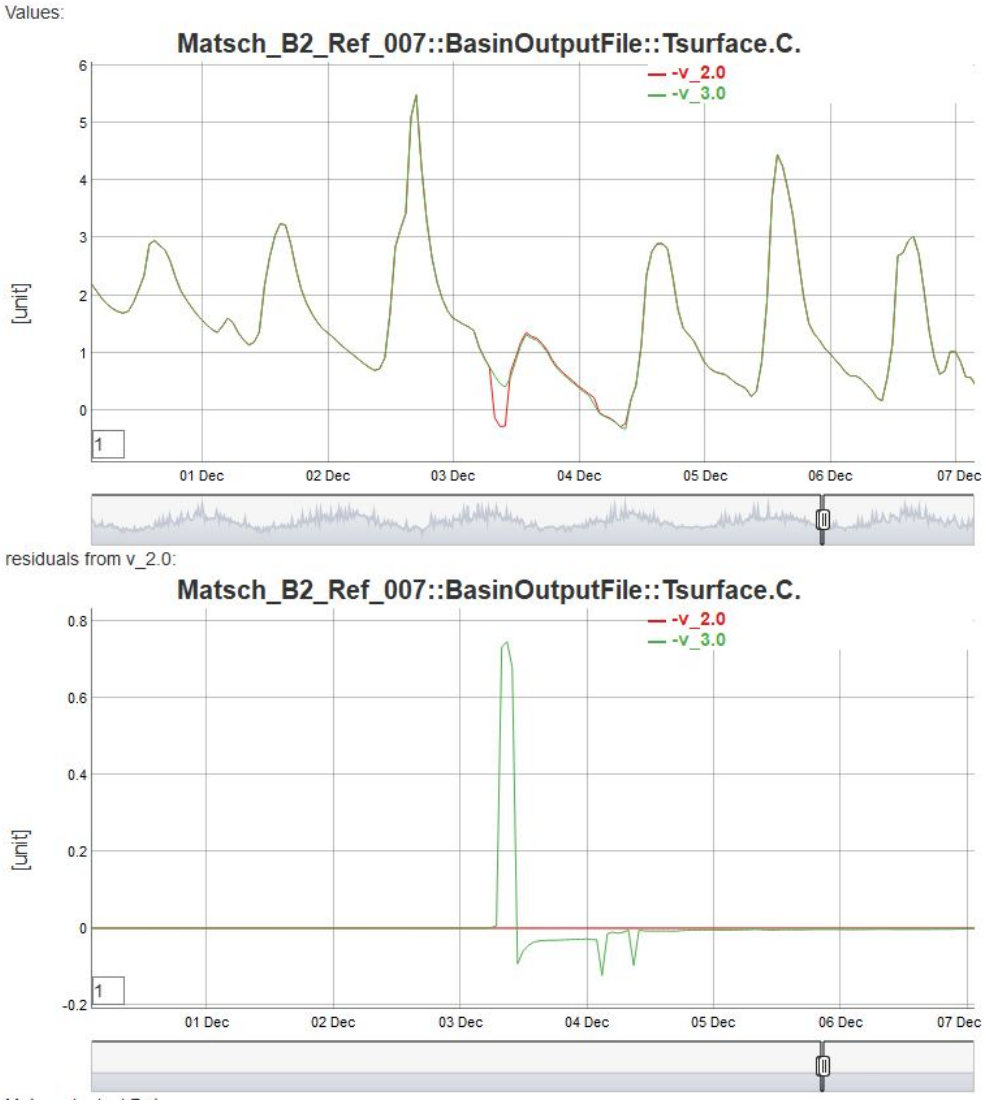


FIGURE 9.1: Examples of different simulated *Tsurface* between v2.0 and v3.0 for B2 test case.

9.1.2 point0001.txt

The significant differences are resumed in Tab. 9.4. The difference are locally meaningful, but they regard a very limited number of time steps (< 0.02 %) or the output time series. For this reason, the results could be considered scientifically valid. An exception is the output variable LObukhovcanopy, which have very large differences for more than the 0.3% of the time series. However, this variable has values that tend to diverge, and it is inherently unstable. This explains how small differences related to numerical approximations can generate large differences in output (Fig. 9.2).

TABLE 9.4: Output variables (file point.txt) whose values are different between v2.0 and v3.0 for B2 test case.

Variable	Max	Mean	Mean_tot	N	Tot	N %
Wind_direction [deg]	11.118	11.118	2.030E-04	1	54767	0.002
Relative_Humidity [-]	0.021	0.021	3.878E-07	1	54767	0.002
Pressure [mbar]	26.604	26.604	4.858E-04	1	54767	0.002
Tsurface [°C]	0.747	0.467	2.605E-05	7	54767	0.013
Tvegetation [°C]	0.388	0.230	2.017E-05	6	54767	0.011
Tcanopyair [°C]	0.390	0.229	2.037E-05	6	54767	0.011
Surface_Energy_balance [W/m2]	3.143	1.865	1.551E-04	5	54767	0.009
Soil_heat_flux [W/m2]	15569.065	1.672	-2.848E-01	9	54767	0.016
Swin [W/m2]	2.897	2.897	5.290E-05	1	54767	0.002
Swdiff [W/m2]	2.844	2.844	5.192E-05	1	54767	0.002
LWin [W.m2]	8.613	8.613	1.573E-04	1	54767	0.002
Lwin min [W/m2]	6.881	6.881	1.256E-04	1	54767	0.002
Lwin max [W/m2]	7.791	7.791	1.422E-04	1	54767	0.002
LWnet [W/m2]	2.358	1.348	-3.111E-05	1	54767	0.002
H [W/m2]	5.113	5.113	1.625E-05	1	54767	0.002
Canopy_fraction [-]	0.031	0.031	5.705E-07	1	54767	0.002
LSAI [m2/m2]	0.125	0.125	2.282E-06	1	54767	0.002
Estored_canopy [W/m2]	3.919	1.847	-8.416E-05	4	54767	0.007
Swv [W/m2]	2.534	2.534	4.691E-05	1	54767	0.002
LWv [W/m2]	1.890	1.518	-8.561E-05	2	54767	0.004
Hv [W/m2]	3.158	1.834	6.462E-05	3	54767	0.005
Hg_veg [W/m2]	5.113	5.113	1.625E-05	1	54767	0.002
Qvegetation [kg/kg]	0.000	0.000	6.445E-09	3	54767	0.005
Qsurface [kg/kg]	0.000	0.000	1.052E-08	5	54767	0.009
Qcanopyair [kg/kg]	0.000	0.000	6.263E-09	2	54767	0.004
LObukhov [m]	**119195.774**	**742.290**	**1.017E+00**	**177**	**54767**	**0.323**
LObukhovcanopy [m]	**51537.448**	**26.045**	**-2.049E+00**	**303**	**54767**	**0.553**
LWup [W/m2]	9.983	4.487	2.688E-04	3	54767	0.005
snow_density [kg/m3]	10232.594	2560.417	-3.716E-01	4	54767	0.007
snow_temperature [°C]	9998.900	9998.900	-3.651E-01	1	54767	0.002
highest_thawed_soil_depth [mm]	350.000	22.500	-3.938E-03	5	54767	0.009

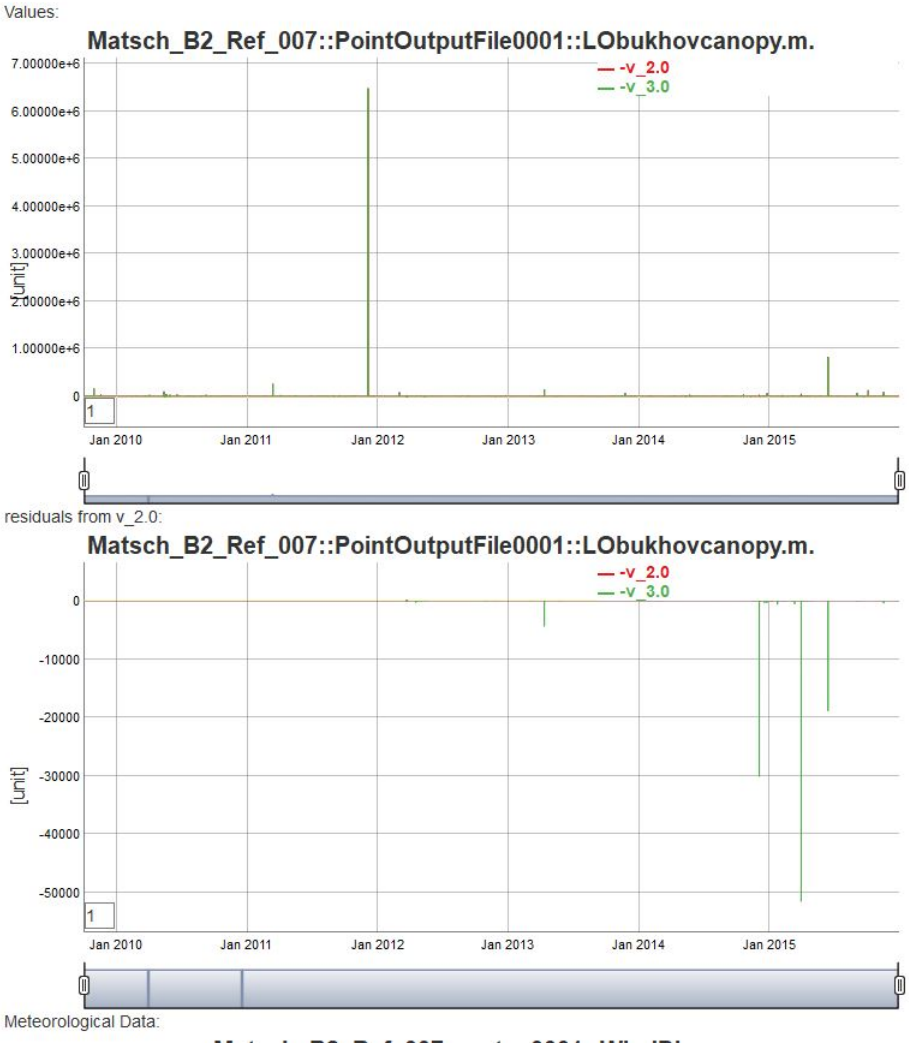


FIGURE 9.2: Example of different simulated *LObukhovcanopy* between v3.0 and v2.0 for B2 test case. The differences are large, but several order of magnitude lower than maximum absolute values of the variable.

9.1.3 psiz0001.txt

The significant differences are resumed in Tab. 9.5. In this case, difference are significant.

TABLE 9.5: Statistics of differences, between v2.0 and v3.0, of the pressure head at variable depth (file psiz.txt) for B2 test case.

Variable	Max	Mean	Mean_tot	N	Tot	N %
X5.0	1343.074	56.141	-6.447E-01	381	54768	0.696
X15.0	1155.217	90.734	-8.967E-02	155	54768	0.283
X25.0	1016.621	69.167	8.959E-02	201	54768	0.367
X40.0	904.699	50.207	4.032E-02	237	54768	0.433
X60.0	481.952	30.290	7.372E-02	325	54768	0.593
X85.0	235.021	22.925	1.891E-01	518	54768	0.946
X125.0	91.391	15.809	3.469E-01	944	54768	1.724
X175.0	90.970	22.697	1.594E-01	349	54768	0.637
X225.0	41.018	18.341	1.448E-01	205	54768	0.374

A graphical representation of such differences is given in Fig. 9.3. It can be noticed that the differences are large, but several order of magnitude lower than maximum absolute values of the variable. Moreover, differences increase in time steps when the models has difficulties in finding convergence. In fact, for some periods the variable is not defined also in the 2.0 version. In particular, it is interesting to observe how the differences are becoming large when *SoilLiqWaterPressProfile* tends to very large negative values. This is a condition when the basis of the power function of Eq. 7.2 tends to 0. In such conditions the approximation of Eq. 7.1 generate significant numerical differences.

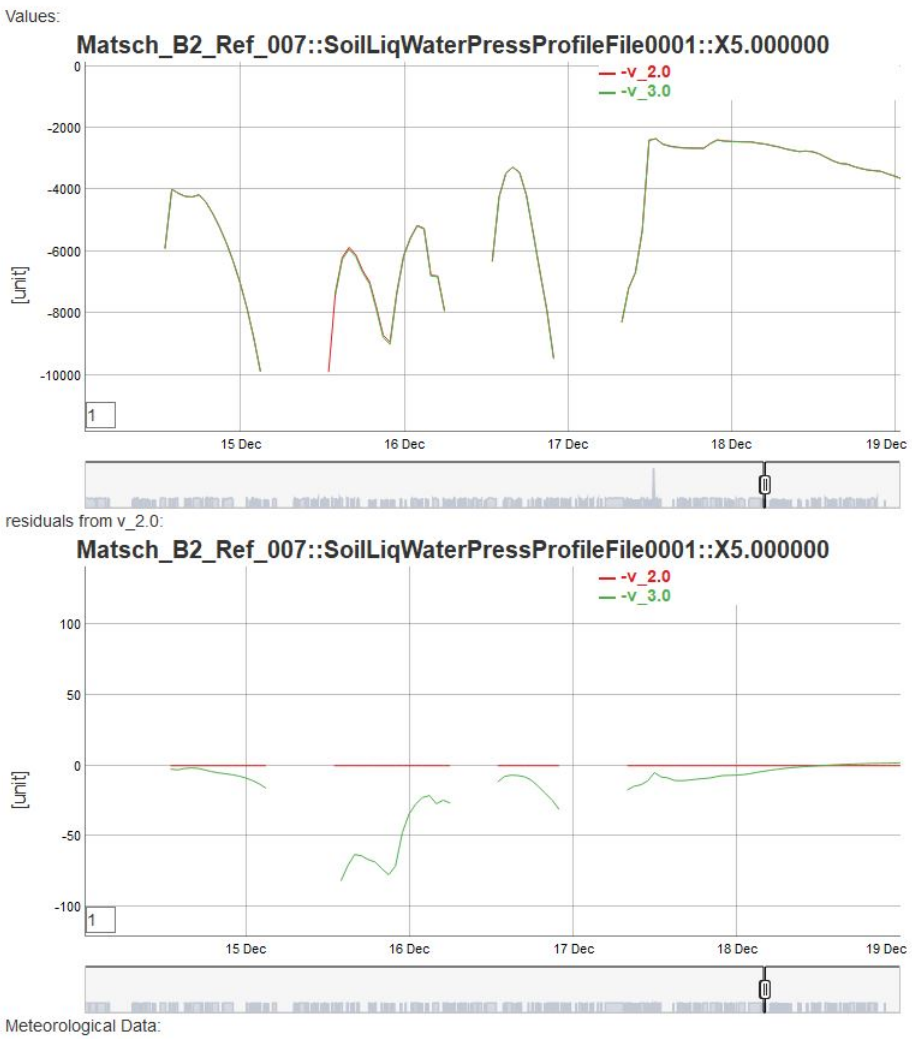


FIGURE 9.3: Differences in the simulated *SoilLiqWaterPressProfile* between v2.0 and v3.0 for B2 test case. The differences are large, but several order of magnitude lower than maximum absolute values of the variable.

9.2 snow (3D test case)

Since this test has a 3D setup, the output are expressed both in terms of time series, whose files can be found in the folder output-tabs, and in terms of maps, whose files are inside output-maps. Differences arose in basin.txt and snowcover.txt (output-tabs) and in snow maps (output-maps). The differences in the time series are limited and similar to the previous 1D test case. Therefore, here, the analysis focused only on maps.

9.2.1 snodepthN*.asc

Differences up to a maximum of +50 - 50 mm. The error value is significant above the experimental error, which can be estimated in about 10 mm. However, the error is for a limited number of pixels (maximum 15 on 6188 computation grid cells, the 0.25%). An example of a snowdepth map is reported in Fig. 9.4.

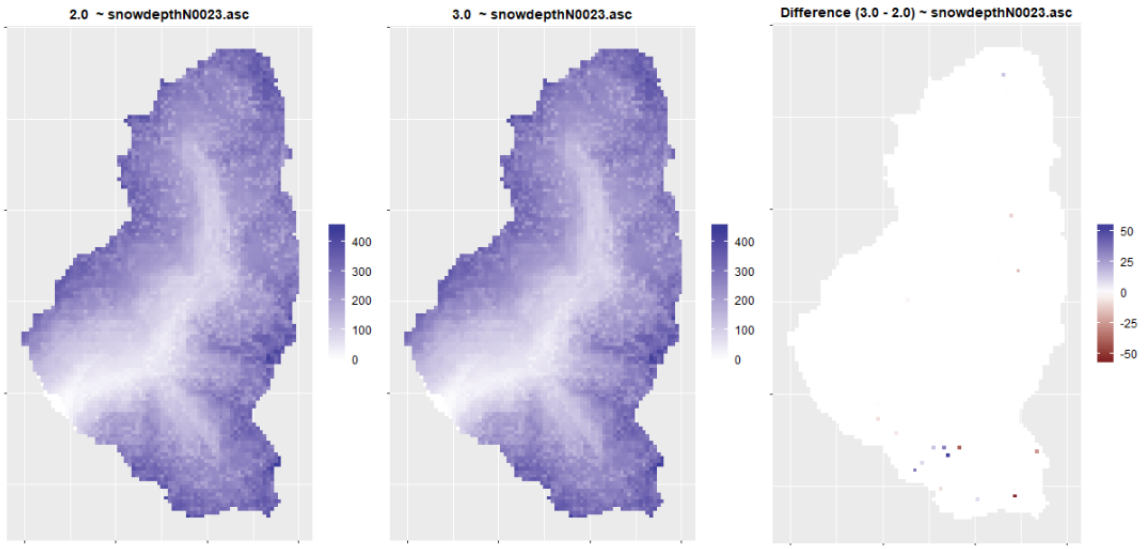


FIGURE 9.4: Example of snowdepth map for snow test case.

9.2.2 snowdurationN00*.asc

Differences up to a maximum 1 day. The error value is not significant and for a limited number of pixels (maximum 3 on 6188 computation grid cells, the 0.05%). An example of a snowduration map is reported in Fig. 9.5.

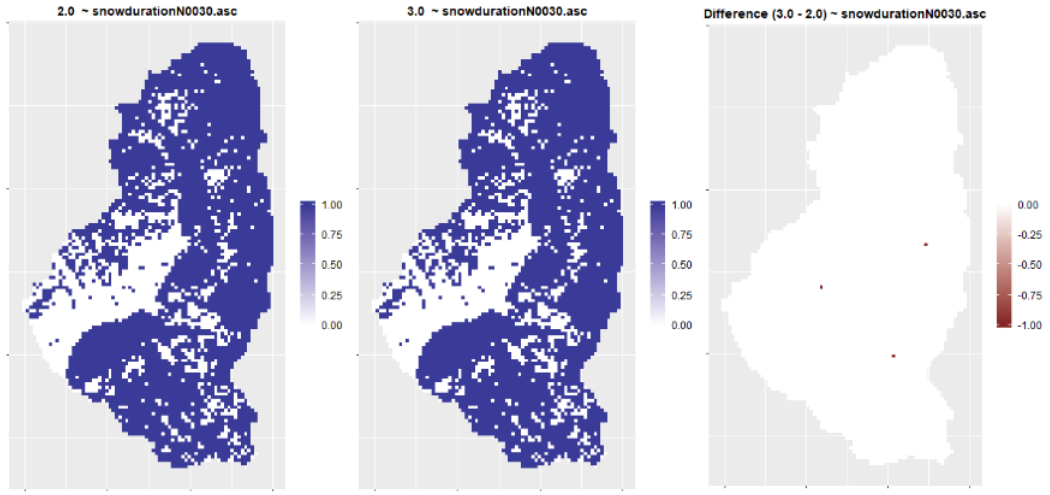


FIGURE 9.5: Example of snowduration map for snow test case.

9.2.3 snowmeltedN000*.asc

Similar results to snowdepth. Differences up to a maximum of +15 - 15 mm. The error value is significant above an experimental error, which can be estimated in about 1 mm. However, the error is for a limited number of pixels (maximum 15 on 6188 computation grid cells, the 0.25%). An example of a snowmelted map is reported in Fig. 9.6.

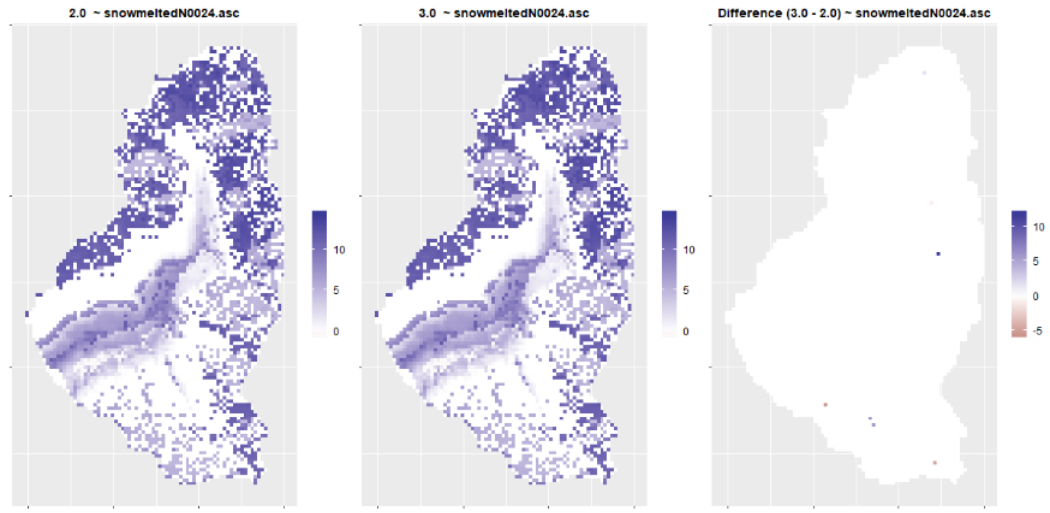


FIGURE 9.6: Example of snowmelted map for snow test case.

9.2.4 snowsublN00*.asc

Similar results to snowdepth. Differences up to a maximum of +0.5 - 0,5 mm. The error value is not significant below an experimental error, which can be estimated in about 1 mm. However, the error is for a limited number of pixels (maximum 15 on 6188 computation grid cells, the 0.25%). An example of a snowduration map is reported in Fig. 9.7.

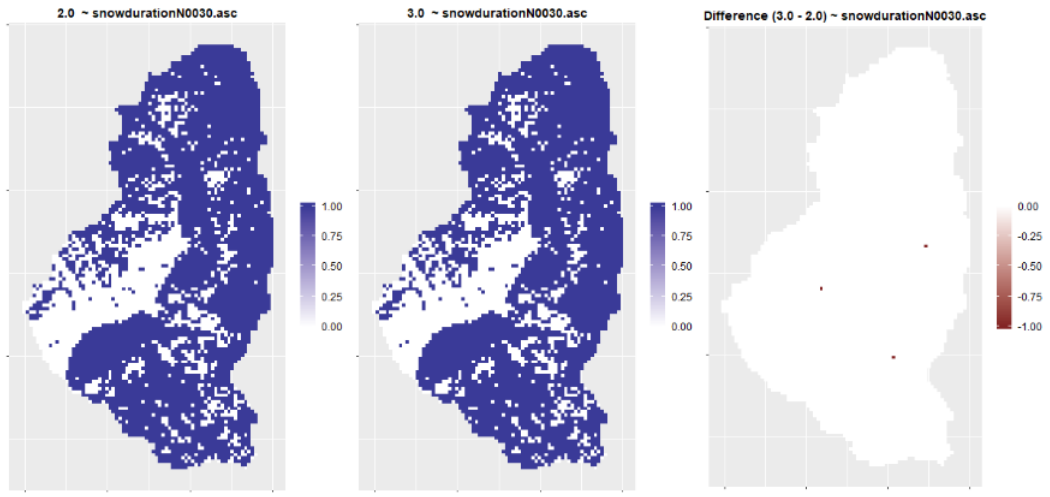


FIGURE 9.7: Example of snowduration map for snow test case.

9.2.5 Concluding remarks on the scientific validation

Locally, the values of some output time steps or of some grid cells have significant differences, higher than the acceptable error for the specific variable. However, the number of cells is very limited ($< 0.5\%$). Therefore, the results of V3.0 could be considered scientifically valid, also for the the tests who fail. In general test failure reveal either a highly not linear or unstable behaviour of the considered output variable, or numerical instability in the original code. Therefore small numerical differences could trigger large output differences in some variables. A scientific validation of other 3D test cases and long simulations is recommended.

Chapter 10

Conclusions and outlook

In this thesis, a reengineering and optimization of the GEOtop software package has been performed. Starting from the 2.0 version, which has considered as benchmark, a new version, called 3.0, has been produced.
The development followed the "best programming practices":

- usage of build tools (Meson and CMake) to automate workflows;
- usage of version control system (Git), to safely save every incremental changes and to allo code reviews;
- code profiling to identify bottlenecks (Likwid-perfctr, Callgrind and the class Timer);
- code re-usage (i.e., in data structures thanks to operator overloading) for modularity and to reduce potential typos;
- code testing thanks to a unit tests framework and exception handling;
- documentation of design and purposes using Doxygen interface (http://www.doxygen.nl/).

The main reengineering efforts on GEOtop 3.0 were the following:

- the code was translated from C to C++;
- the old data structures were replaced by new ones using more modern, object oriented, approach to optimize memory access;
- a code profiling was performed over three test cases representative of the different operational model usage conditions;
- on the basis of profiling results, several optimizations have been performed: some computationally expensive mathematical operations were replaced by less CPU-expensive expressions, some parts of the code were parallelized using OpenMP, compiling vectorization flags were introduced;
- optimization results have been analyzed and discussed. Math optimization resulted in a significant improvement for all test cases up to 30% of CPU time. Benefit of vectorization were limited, while OpenMP

optimization produced an overhead in the 1D test case, while profduced a limited improvement fot the 3D test cases, up to a 20%. Also scalability was limited; this is due to the fact that the parallelized functions cover a limited part of the code.

- it was found that locally, the values of some output time steps or of some grid cells have significant differences, higher than the acceptable error for the specific variable. However, the number of cells is very limited ($< 0.5\%$). Therefore, the results of v3.0 could be considered scientifically valid, also for the failing tests.

To conclude, now the GEOtop v3.0 has an optimized code with more modern structures and a rigorous validation. This can become a basis for the following further improvements that could make GEOtop more HPC efficient:

- **parallelization**: with OpenMP, extending the parallelized part to thread safe functions inside the time loop, and eventually re-writing them to avoid conflicts among threads and/or with MPI, using domain decomposition so that the all work could be divided among process, resulting in a noticeable run time decrease;

- **I/O**: improving the input-output data flow to make the usage of external libraries for the input data (i.e., NetCDF (https://www.unidata.ucar.edu/software/netcdf/) and Meteoio) more feasible;

- **maths optimization**: using libraries like BLAS (http://www.netlib.org/blas/) or Eigen (http://eigen.tuxfamily.org/index.php?title=MainPage); if whole the code will be parallelized, Plasma and Magma could be chosen (http://prace.it4i.cz/PlasMagma-09-2018);

- **scientific validation**: some specific est failure reveal numerical instability in the original GEOtop v2.0 code. A revision of the code numeric and a scientific validation of other 3D test cases and long simulations is recommended.

Bibliography

Armijo, Larry (1966). "Minimization of functions having Lipschitz continuous first partial derivatives". In: *Pacific Journal of Mathematics* 16.1. DOI: 10.2140/pjm.1966.16.1.

Bavay, M. and T. Egger (2014). "MeteoIO 2.4.2: A preprocessing library for meteorological data". In: *Geoscientific Model Development* 7.6. ISSN: 19919603. DOI: 10.5194/gmd-7-3135-2014.

Bertoldi, Giacomo (2004). "The water and energy balance at basin scale: a distributed modeling approach". PhD thesis. ISBN: 8884430690.

Bortoli, Elisa (2017). "Modelling soil moisture dynamics in Alpine Grasslands". PhD thesis.

Dall'Amico, M, S Endrizzi, and S Tasin (2018). "MYSNOWMAPS : OPERATIVE HIGH-RESOLUTION REAL-TIME SNOW MAPPING". In: *Proceedings, International Snow Science Workshop, Innsbruck, Austria, 2018*, pp. 328–332.

Della Chiesa, S et al. (2014). "Modelling changes in grassland hydrological cycling along an elevational gradient in the Alps". In: *Ecohydrology* 7.6, n/a–n/a. ISSN: 1936-0592. DOI: 10.1002/eco.1471. URL: http://dx.doi.org/10.1002/eco.1471.

El-Mikkawy, Moawwad and Abdelrahman Karawia (2006). "Inversion of general tridiagonal matrices". In: *Elsevier* 19, pp. 712–720. DOI: 10.1016/j.aml.2005.11.012.

Endrizzi, S. et al. (2011). *GEOtop Users Manual*. Tech. rep. July, p. 130.

Endrizzi, S. et al. (2014). "GEOtop 2.0: Simulating the combined energy and water balance at and below the land surface accounting for soil freezing, snow cover and terrain effects". In: *Geoscientific Model Development* 7.6, pp. 2831–2857. ISSN: 19919603. DOI: 10.5194/gmd-7-2831-2014.

Endrizzi, S. et al. (2017). *GEOtop Users Manual*.

Engel, Michael et al. (2017). "Snow model sensitivity analysis to understand spatial and temporal snow dynamics in a high-elevation catchment". In: *Hydrological Processes* 31.23, pp. 4151–4168. ISSN: 10991085. DOI: 10.1002/hyp.11314.

Genuchten, M.T. van (1980). "A Closed-form Equation for Predicting the Hydraulic Conductivity of Unsaturated Soils". In: *Soil Sci. Soc. Am. J.* 44, pp. 892–898.

Gueymard, Christian (1989). "An atmospheric transmittance model for the calculation of the clear sky beam, diffuse and global photosynthetically active radiation". In: *Agricultural and Forest Meteorology* 45.3-4, pp. 215–229. ISSN: 01681923. DOI: 10.1016/0168-1923(89)90045-2.

Heaton, Dustin and Jeffrey C. Carver (2015). "Claims about the use of software engineering practices in science: A systematic literature review". In:

Information and Software Technology 67, pp. 207–219. ISSN: 09505849. DOI: 10.1016/j.infsof.2015.07.011. URL: http://dx.doi.org/10.1016/j.infsof.2015.07.011.

Horgan, Joseph R., Saul London, and Michael R. Lyu (1994). "Achieving Software Quality with Testing Coverage Measures". In: *Computer* 27.9, pp. 60–69. ISSN: 00189162. DOI: 10.1109/2.312032.

Iqbal, Muhammad (1983). *Introduction to Solar Radiation*.

Kelley, Carl T. (2003). *Solving nonlinear equations with Newton's method*. Siam.

Liston, Glen E. and Kelly Elder (2006). "A Meteorological Distribution System for High-Resolution Terrestrial Modeling (MicroMet)". In: *Journal of Hydrometeorology*, pp. 217–234. DOI: https://doi.org/10.1175/JHM486.1.

Niessner, H (1983). "On computing the inverse of a sparse matrix". In: *International Journal for Numerical Methods in Engineering* 19.August 1981, pp. 1513–1526. DOI: https://doi.org/10.1002/nme.1620191009.

Orlandini, Stefano et al. (2003). "Path-based methods for the determination of nondispersive drainage directions in grid-based digital elevation models". In: *Water Resources Research* 39.6, pp. 1–8. DOI: 10.1029/2002WR001639.

Rigon, Riccardo, Giacomo Bertoldi, and Thomas M. Over (2006). "GEOtop: A Distributed Hydrological Model with Coupled Water and Energy Budgets". In: *Journal of Hydrometeorology* 7.3, pp. 371–388. ISSN: 1525-755X. DOI: 10.1175/JHM497.1.

Stroustrup, Bjarne (2013). *The C++ Programming Language : Fourth Edition*. ISBN: 0321154916. URL: https://books.google.de/books?id=LgmqCAAAQBAJ.

Van Der Vorst, H. A. (1992). "BI-CGSTAB: a fast and smoothly converging variant of BI-CG for teh solution of non symmetric linear systems". In: *Society for Industrial and Applied Mathematics* 13.2, pp. 631–644. DOI: https://doi.org/10.1137/0913035.

Wilson, Greg et al. (2014). "Best Practices for Scientific Computing". In: *PLoS Biology* 12.1. ISSN: 15449173. DOI: 10.1371/journal.pbio.1001745. arXiv: arXiv:1210.0530v4.