PLEASE NOTE THAT THIS IS NOW OBSOLETE! DO NOT USE THIS MANUAL

Alpine3D Contributers

January 31, 2017

Contents

Chapter 1

Introduction

1.1 Overview

Alpine3D is a spatially distributed, land-surface type model comprising several phyically and conceptually submodules: a snowpack model (SNOWPACK), a radiation balance model, a runoff model, and a snow transport model. Additionally, it provides some routines for interpolation of meteorological data. A brief overview of Alpine3D and its application to hydrological issues can be found in in [2].

Alpine3D is written in C (Snowpack), C++(energy balance, Snowdrift) and Fortran (runoff, interpolation) on Unix/Linux. It exploits the GRID technology POPC++ [1] for parallelizing individual submodules. To use the parallel capabilities of Alpine3D it must be installed together with POPC++. In a multiuser environment it is recommended to install additionally a GRID resource management system like the Globus Toolkit [5].

Chapter 2

Installation

2.1 System Requirements

Alpine3D is a Linux application and in its sequential mode it can be installed on every (?) Unix/Linux system. Presently it can be used in its parallel mode only on the Linux cluster at SLF and on the HPC cluster "Zeus" at WSL.

2.2 Source code

The Alpine3D source code is maintained with SVN. For a good online documentation see [6], and the basic usage of the `svn` is also briefly described in Ch. 6. The repository can be accessed (only locally at SLF) via the URL `svn://svn.slf.local/alpine3d` after receiving a password from the IT team. To check out a working copy of Alpine3D use the svn command

```
svn co svn://svn.slf.local/alpine3d/alpine3d/trunk
```

in a terminal. This will create a subdirectory `trunk` in your *present* working directory. In the `trunk` directory you will find the following files and subdirectories of Alpine3D:

- `./main/` (main application class)
- `./common/` (array classes, data marshalling functions from POPC++)
- `./ebalance/` (energy balance class, radiation model, view factors etc)
- `./snowdrift/` (snowdrift class: saltation, suspension)
- `./snowdrift_par/` (parallel snowdrift implementation)
- `./snowpack/` (snowpack class, core files of SNOWPACK)
- `./inout/` (i/o classes)
- `./runoff/` (runoff functions, FORTRAN!)
- `./interpol/` (interpolation routines for 2d meteo input, FORTRAN!)
- `./deploy/` (directory for popc objects)
- `./Interface/` (a gui for Alpine3D)

`./doc/` (this manual, documentation)
`./tools/` (some auxiliary scripts)
`./current_snow/` (input directory, should be deleted from svn!)
`run_dischma_seq.sh` (an example start script for sequential runs)
`run_dischma_par.sh` (an example start script for parallel runs)
`Makefile` (the top level Makefile)
`Makefile.par` (the top level Makefile only for parallel drift)

Note that the program is continously improved and corrected. Make sure that you are up to date by checking the status of your working copy and update if necessary by typing

```
svn up .
```

in your `trunk`-directory (see also Ch. 6 for details)

2.3 Environment

Using compilers, POPC++, and GT requires you to have your environment set up correctly. At SLF, add the following lines to an appropriate dot-file, i.e. to `.profile` or `.bash_profile` in your home directory.

```
#-- globus ---------------------------------------------------
export GLOBUS_LOCATION=/usr/local/gt3/gt3.2
source $GLOBUS_LOCATION/etc/globus-user-env.sh
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/gt3/gt3.2/lib

#-- popc -----------------------------------------------------
# Note : Var names changed from PAROC_ to POPC_ in POP-C++-1.3
export POPC_LOCATION=/usr/local/popc
export PATH=$PATH:/usr/local/popc/bin
export POPC_JOBSERVICE=grid1.slf.local:2711

#-- fortran --------------------------------------------------
export PATH=$PATH:/usr/local/intel_fc_81/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/intel_fc_81/lib
```

and source the file. Before complaining, make sure that everything is set correctly (i.e. verify your environment by typing `set` in a terminal).

2.4 Compilation

2.4.1 Making at SLF

Compiling Alpine3D requires a C, C++ and a FORTRAN90 compiler. At SLF, gcc, g++ and ifort (Intel compiler) is used. Compilation is done using `make` with the top level Makefile: Use

```
make all
```

in your trunk directory to generate the `Alpine3D` executable for sequential computation. In order to generate the executable `Alpine3D.popc` for parallel computing use

```
make all_par
```

and subsequently

```
make deploy_par
```

to move the popc-modules to the `./deploy` directory and create the object description file `./deploy/objectmap.conf` required by POPC++. For making the parallel snowdrift implementation see Ch. 7.

2.4.2 Specifics on Zeus

A brief summary of the HPC cluster Zeus at WSL is given below. You can login to zeus@wsl.ch after receiving a password from Markus Reinhard (`markus.reinhardt@wsl.ch`)

- CPUs: AMD Opteron Dual Core 64 Bit processors. Each unit has two Dual CPUs and the cluster comprises 16 nodes (ip-adresses 172.16.2.1-16) which makes 16 x 2 x 2 = 64 cores
- Memory: Each unit has 8 GByte RAM.
- Netork: The units are connnected by a Gigabyte Ethernet via switch. Additionally, compute nodes are connected by Myrinet. The transfer rate is 900 MBytes/s (r+w).
- OS: Suse Linux 9.3
- Software: Sun Grid Engine (sge), ls-dyna, globus, mpich, petsc, pathscale (Compiler), R, IDL, ganglia, nagios

Moving the program to Zeus or any other computer requires to modify compilers and libraries in the top level makefile `Makefile`. On Zeus, gcc, g++ and the pathscale Fortran compiler pathf90 can be used. Besides updating your environment accordingly the top level makefile `Makefile` has to be changed: Make sure that the respective linker flag

```
FLIBS=-L/usr/local/intel_fc_80/lib -limf -lifcore
```

valid at SLF is replaced by the appropriate pathscale flag

```
FLIBS=-L/opt/pathscale/lib/2.2 -lmpath -lpathfortran
```

In addition, make sure to add the flag `-fno-second-underscore` to `FFLAGS`. Then go on with making as described in the previous Section.

Note: Sometimes nothing happens when Alpine3D.popc is started on Zeus. Try to open the `./deploy/objectmap.conf` file and remove the `exports` directory in the object description, i.e. replace `/exports/home/...` by `/home/...`.

It is also possible that the job manager unexpectedly dies on Zeus. In order to restart it, login as popc and run the command `popc-1.0-inst/sbin/SXXpopc start`.

Chapter 3

Running Alpine3D

3.1 Testing Alpine3D

After successful compilation, you can immediately test the sequential and the parallel version. By simply typing `Alpine3D` it will print a usage-message and you will get a list of possible command line switches with explanations.

An example call to Alpine3D with appropriately set command line switches can be found in the example shell scripts `run_dischma_*.sh` in your `trunk` directory. These scripts start examples start your local version of Alpine3D with input data for the Dischma domain in the directory
`/usr/local/org/FE/SCHNEEP/SMM/projects/alpine3d-testcases/dischma`
The input data sets up a simulation of the Dischma catchment with 2d meteorological input. Unfortunately, this directory is presently only accessible for the team "Snow-cover and Micrometeorology".

The output and the logfile `stdouterr.log` is written to
`/usr/local/org/FE/SCHNEEP/SMM/projects/alpine3d-testcases/dischma/output/test`
Output fields are compatible with the Ascii-ArcInfo grid format. All files follow the naming convention `<julian-day>.<extension>` where extensions are used as given in Tab. 3.1

.lwr	long wave radiation
.swr	short wave radiation
.sdp	snowdepth
.alb	albedo
.swe	snow water equivalent
.tss	snow surface temperature
.sca	snow covered area

Table 3.1: File name extension in Alpine3D

You can open these output files, e.g. with the Alpine3D GUI by switching to `Interface` subdirectory, typing `view.bat` and opening one of the files in that directory.

3.2 Setting up a simulation

Setting up your own simulation can be done by preparing input data and adjusting the parameters and paths in the example start scripts. Note, presently there are two (three) different files where parameters and paths have to be adjusted if you want to do simulations with 1d (2d) meteorological input. These are

- The start script.
- The snowpack parameter file specified by the -snowparam switch in the start script
- The runoff/interpolation parameter file specified by the -meteopath switch in the start script (only for 2d meteo input)

Make sure to have all paths in all files set correctly.

Chapter 4

Input Data

4.1 General

To be filled: Directories, formatting, time stamps, etc

4.2 Meteorological input

To be filled. Also ask for [7]. Even in case 2d meteorological input is applied, some attention has to be paid concerning the remaining 1d meteorological input. Therefore, better read the 1d meteorological input section too.

4.2.1 1d meteorological input

- No time gaps are possible; make sure the gaps are eliminated by extrapolation before starting Alpine3d since the sun position in the radiation balance module is always computed from the expected date following the main imposed Alpine3D time step.
- Starting time and time steps should be consistent between 1d- and 2d meteorological input.
- The radiation input (global shortwave and longwave radiation or cloud cover fraction) **has** to be measured at an **exposed** measurement site.
- Latitude, Longitude in the header of 1d meteorological file should be the coordinates of the center of the model domain; Swiss Coordinates and height in the header of 1d meteorological file should be the coordinates and height of the radiation input station.
- All meteorological input has to be in winter time (e.g.Davos: UTC+1) (i.e. no summer time is accounted for).

4.2.2 2d meteorological input

Chapter 5

ToDo

5.1 Model problems

- The model overestimates snowdepth in low elevations and underestimates it in higher elevations.

5.2 Implementation problems

- Reorganization (i.e. simplification) of the specification of parameters and paths (e.g.: each module should come up with its own parameter class `<module>Param.cc/h/ph` and paths can be set only in the start script)
- Improve portability (i.e. GNUization with the configure/autoconf/automake mechanism)
- Some binary output is generated by one of the Fortran codes and written to stdout or stderr and hence to the logfile. This prevents to search the logfile with grep. ->Incovenient.

Chapter 6

Subversion in a nutshell

6.1 RTFM

Documentation can be found on the website
`http://subversion.tigris.org/`
and in the online book (sectioned html)
`http://svnbook.red-bean.com/`
which nicely tells you what it is all about:
If C gives you enough rope to hang yourself, think of subversion as a sort of rope storage facility (from the preface)

6.2 Before you start

- The repository at SLF has the (url) location
  `svn://svn.slf.local/alpine3d/alpine3d/trunk`.
  If you prefer access to the svn via command line it might be convenient to to set
  `export SVN_ROOT="svn://svn.slf.local/alpine3d/alpine3d/trunk"`
  There are also nice and more convenient x-applications for accessing the repository. Check it out.
- If you even fail at *closing* old-school vi without the reference card (like me) better
  `export EDITOR=emacs`
  in your shell before committing something via command line.

6.3 Basic work cycle

- Initial checkout:

`svn checkout $SVN_ROOT` (alias: `svn co $SVN_ROOT`)

creates a working copy of the repository by copying the complete directory tree (including the directory `trunk`) from the repository to your present working directory. Additionally `.svn` subdirectories are created in each subdirectory which hold status information about your working copy.

- Updating the working copy with the most recent version. Type

  `svn update` (alias: `svn up`)

  in your `trunk` directory. If you want to update your working copy with an older (less buggy?) version use

  `svn up --revision < revision no >`

  You can also update only particular directories/files of your working copy by using

  `svn up <filename>`

- Making changes to your working copy. Change your working copy by by moving, copying, deleting or adding files. Use
  `svn add, svn delete, svn copy, svn move` These changes take place immediately in your working copy and the repository is changed after your next `commit`. Note: if you simply invoke system commands, e.g. `rm`, to delete a file, the deletion is not scheduled as a future change in the repository and you might receive an error at next `commit`.

- Checking the status of the working copy:

  `svn status`

  reports *if* there are changes in your files after your last update. If you want to know *what* has changed use

  `svn diff`

  If you want to suppress status information about your messy working with bunch of additional files use the `-q` option. If you want to additionally have information if there are more recent versions in the repository use

  `svn status -qu`

  The first column of the output contains flags indicating the status of the file, the most important is a capital "C": Conflict, then your version won't compile. "M" means modified, "A" added and "D" deleted. If a file has a "*" in the second column, a more recent version of that file is available in the repository.

- Undo:

  `svn revert alpine3dFile.cc`

  recovers the state of the file at previous update.

- Commit your changes:

  ```
  svn commit
  ```

  If not invoked with the "-m" switch the default editor will open for entering comments on your changes.

- Examining the revision history of the project:

  ```
  svn log
  ```

  By passing a filename, the history of only this file is shown. If you want to make sure to read the complete history of alpine3d-changes then use

  ```
  svn log svn://svn.slf.local/alpine3d/alpine3d/trunk
  ```

  All missing revision increments stem from changes in snowpack which is maintained in the same repository in

  ```
  svn://svn.slf.local/alpine3d/snowpack/
  ```

  If you really want to read *every commit message of the whole repository*, type

  ```
  svn log svn://svn.slf.local/alpine3d
  ```

6.4 Rules

Needless to say:

1. Don't commit without comment.
2. Don't commit if it doesn't compile.

Chapter 7

Snowdrift module

7.1 Overview

The following description refers to the parallel version of the snowdrift module located in `snowdrift_par`. From the computational point of view it is identical to the sequential version at the date where this directory was committed (20. December 2007). But in the parallel implementation in `snowdrift_par` all redundant routines have been removed and the naming of functions and variables has been made more intuitive.

The snowdrift module comprises three different classes. The base class `SnowDrift` (Interface in `SnowDrift.h` and `SnowDrift.ph` and the derived classes `SnowDriftParallel` (Interface in `SnowDriftParallel.h` and `SnowDriftParallel.ph`) and `SnowDriftWorker` (Interface also in `SnowDriftParallel.h` and `SnowDriftParallel.ph`) The implementation of the `SnowDriftParallel` class can be found exclusively in `SnowDriftParallel.cc` whereas the implementation of `SnowDrift` and `SnowDriftWorker` is distributed over the files

```
SnowDrift.cc
Suspension.cc (contains solely the SnowDriftWorker::SubSuspension method)
SnowDriftFEInit (contains initialization routines for the FE method)
SnowDriftFENumerics (contains numerical element routines for the FE metho
SnowDriftFEControl (contains additional FE routines for SnowDrift and
SnowDriftWorker)
```

The remaining files

```
PackSnowDrift.cc
PackSnowDriftWorker.cc
checksum.cc
marshal_drift.cc
```

are required for PopC++ and finally `clock.h/cc` contains a simple class for time measurements.

For a basic understanding of the computation flow of the snowdrift model check the method `SnowDrift::Compute()` in the file `SnowDrift.cc` which consists basically of a call to the Saltation and the Suspension routine (besides some initialization and io-operations).

For the Saltation you have to dig into the file `Saltation.cc`. Some of the routines required there are still located in the `snowpack_core` subdirectory of the repository in the file `./snowpack/snowpack_core/Saltation.c`

For the suspension the main sequence of actions can be found in `SnowDriftWorker::SubSuspension()` in the file `Suspension.cc` with self-explenatory naming of the routines.

7.2 Details of the implementation of the Finite Element method

All theoretical preliminaries can be found in Marcs report [4]. As explained there, linear, hexahedral elements are used for the FE method. In the following I give further details about the location of individual procedures of the FE method.

7.2.1 Isoperimetric transformation

The nodes in the reference Element $\tilde{K}$ have coordinates

$$\begin{aligned} P^{(1)} &= (1,-1,-1) & P^{(5)} &= (1,-1,1) \\ P^{(2)} &= (1,1,-1) & P^{(6)} &= (1,1,1) \\ P^{(3)} &= (-1,1,-1) & P^{(7)} &= (-1,1,1) \\ P^{(4)} &= (-1,-1,-1) & P^{(8)} &= (-1,-1,1) \end{aligned} \tag{7.1}$$

Note that the coordinate system in the reference element is different from global coordinate system (cf. Fig. 7.1. The 8-node trilinear hexahedron basisfunctions defined on the reference element K are explicitely given by

$$\phi^{(k)}(\xi) = \frac{1}{8}\prod_{\alpha=1}^{3}(1+\xi_\alpha P_\alpha^{(k)}), \qquad k = 1\ldots 8 \tag{7.2}$$

The implementation can be found in `computePhi()` in `SnowDriftFEnumerics.cc`.

In terms of the ϕ^k, the isoperimetric transformation $F : \tilde{K} \to K$ reads

$$F_\alpha(\xi) = \sum_{i=1}^{8} Q_\alpha^{(i)}\,\phi^{(i)}(\xi) \tag{7.3}$$

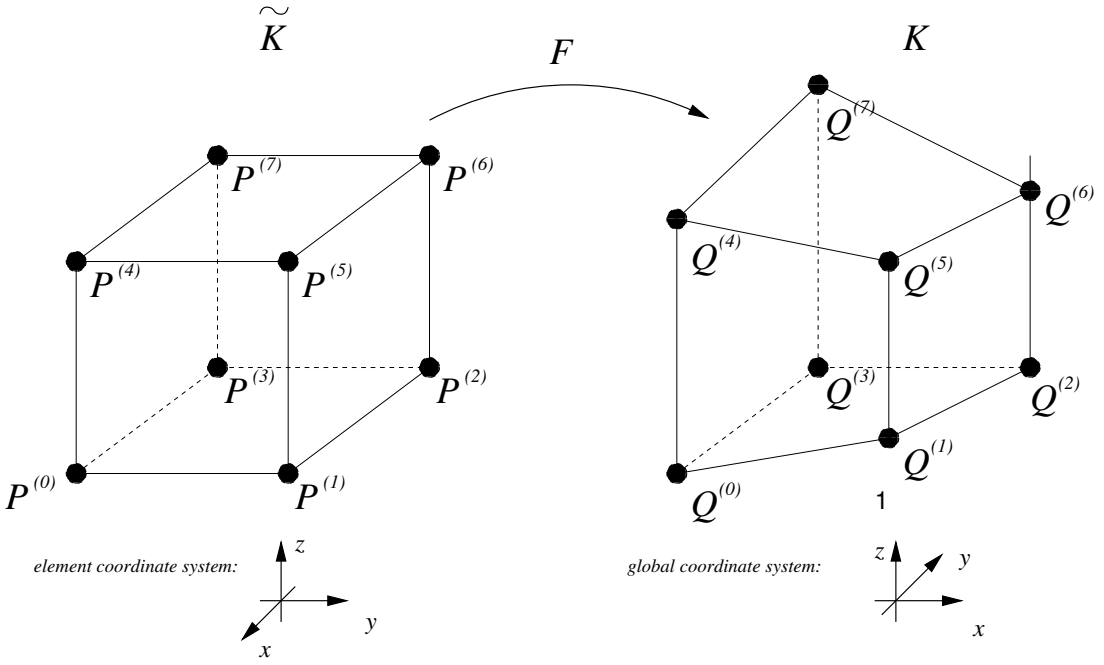

Figure 7.1: Enumeration and coordinate systems in the reference and original element respectively

and hence the Jacobian of the transformation is given by

$$J_{\alpha,\beta}(\xi) = \frac{\partial F_\alpha(\xi)}{\partial \xi_\beta} = \sum_{k=1}^{8} Q_\alpha^{(k)} \frac{\partial \phi^{(k)}(\xi)}{\partial \xi_\beta} \tag{7.4}$$

which is implemented in `computeJacobian` in `SnowDriftFEnumerics.cc`. The gradients of ϕ^k required for the jacobian are computed in `computeGradPhi()` in `SnowDriftFEnumerics.cc`.

7.2.2 Integrals over elements

The integrals over elements are performed by integrating over reference elements. The weak solution of involves integrals over the original elmenents K. The integration is performed by transforming to the reference element $\tilde{K}$ via F^{-1}. Accordingly, integrals are computed over $\tilde{K} = F^{-1}(K)$ with the transformation formula

$$\int_K \mathrm{d}^3x \; f(x) = \int_{\tilde{K}} \mathrm{d}^3\xi \; |\det(J(\xi))| \, f(F(\xi)) \tag{7.5}$$

The integrals over reference elements are done by a two-point (per dimension) Gaussian quadrature with points $\pm 1/\sqrt{3} \approx 0.57735$. These points are set in `setQuadarturePoints` in `SnowDriftFEnumerics.cc`.

Since some integrals over elements involve the inverse of the jacobian J (see Eq. in [4]) it is advantageous it in terms of the adjugate of J (denoted by J_0 in [4]. The inverse of J and the adjugate $\mathrm{adj}(J) =$ are related by

$$J^{-1} = \det(J)^{-1}\mathrm{adj}(J) \tag{7.6}$$

This matrix is computed in `computeAdjugateMatrix` in `SnowDriftFEnumerics.cc`.

7.2.3 Boundary conditions

As opposed to [4] boundary conditions are now implemented as weak Robin boundary conditions which has the advantage of higher flexibility.

Additional element contributions

The derivation of the weak formulation of Eq. (25) in [4] (i.e. by multiplying with a test function v and integrating by parts) in general leads to an additional boundary term, viz

$$-\int_\Omega \mathrm{d}x\, \big[\nabla\cdot(K(x)\nabla c(x))\big]v(x) \;=\; \int_\Omega \mathrm{d}x\, \big[K(x)\nabla c(x)\big]\cdot\nabla v(x) \tag{7.7}$$

$$-\int_{\partial\Omega} \mathrm{d}a\, n(x)\cdot\big[K(x)\nabla c(x)\big]v(x)\;, \tag{7.8}$$

where $n(x)$ is the *outward* normal vector field on the boundary $\partial\Omega$ of the domain Ω.

The most general Robin boundary condition (also: generalized Neumann or flux boundary condition) can be defined in the form

$$-n(x)\cdot\big[K(x)\nabla c(x)\big] = \gamma(x)\big(c(x)-g_{\mathrm{D}}(x)\big)+g_{\mathrm{N}}(x) \tag{7.9}$$

in terms of functions $\gamma, g_{\mathrm{D}}, g_{\mathrm{N}}$ which are defined on the boundary $\partial\Omega$. The limiting case of pure Dirichlet conditions $c(x) = g_{\mathrm{D}}(x)$ can be formally obtained by $\gamma(x)\to\infty$ and pure Neumann conditions $-n(x)\cdot\big[K(x)\nabla c(x)\big] = g_{\mathrm{N}}$ by setting $\gamma(x) = 0$.

Consequently, the weak imposition of the Robin boundary condition (7.9) in Eq. 25 of [4] can be achieved by inserting (7.9) into (7.7) which leads to two additional terms in the weak formulation (27) in [4]: An additional contribution of the load on the rhs of Eq. (27) in [4] is given by (counting minus-signs correctly)

$$b_r(v) := \int_{\partial\Omega} \mathrm{d}a\, \big[\gamma(x)g_{\mathrm{D}}(x) - g_{\mathrm{N}}(x)\big]v(x) \tag{7.10}$$

And the additional contribution from the Robin condition to the element matrix on the lhs of Eq. (27) in [4] reads

$$b_l(c,v) := \int_{\partial\Omega} \mathrm{d}a\, \gamma(x)c(x)v(x) \tag{7.11}$$

Note, to obtain the consistently stabilized conterpart of Eq. (31) in [4] with the Robin boundary conditions, both additional terms (7.10,7.11) have to be evaluated with stabilized test functions.

Integrals over element surfaces

The respective element contributions of the surface integrals (7.10,7.11) are also computed by transformation on the reference element $\tilde{K}$. The isoperimetric transformation F in 7.3 induces six mappings $G_{\alpha,\sigma}$ from the surfaces

$$\tilde{A}_{\alpha,\sigma} = \{\xi \in \tilde{K} : \xi_\alpha = \sigma,\ \sigma = \pm 1,\ \alpha = 1,2,3\} \tag{7.12}$$

of the reference element $\tilde{K}$ to the surfaces of the $A_{\alpha,\sigma}$ of K given by the restriction of F on the respective surface $G_{\alpha,\sigma} := F|_{\tilde{A}_{\alpha,\sigma}}$

Then, the usual transformation formula for surface integrals applies to the surfaces of the element K via

$$\int_{A_{\alpha,\sigma}} \mathrm{d}a f(x,y,z) = \int_{\tilde{A}_{\alpha,\sigma}} \mathrm{d}\xi_\alpha \mathrm{d}\xi_\gamma \, \|\partial_{\xi_\beta} F(\xi) \times \partial_{\xi_\gamma} F(\xi)\| \Big|_{\xi_\alpha=\sigma} f\big(F(\xi)\big)\Big|_{\xi_\alpha=\sigma} \tag{7.13}$$

and α, β, γ chosen cyclic in $1, 2, 3$. The surface jacobian can be expressed in terms of the adjugate $\tilde{J}$ of the jacobian via

$$\|\partial_{\xi_\beta} F(\xi) \times \partial_{\xi_\gamma} F(\xi)\| = \Big(\sum_\beta \Big[\tilde{J}_{\beta,\alpha}\Big]^2 \Big)^{1/2} \tag{7.14}$$

The application of Robin boundary conditions is done in the method `SnowDriftWorker::applyRobinBC` in `SnowDriftFEControl.cc`. The functions $\gamma, g_\mathrm{D}, g_\mathrm{N}$ can be specified in `SnowDrift::InitializeSystem` in `SnowDriftFEInit.cc`.

7.2.4 Testing the numerics

By choosing the diffusion $K = \mathrm{diag}(0, 0, K_z)$, the flow $\mathbf{u} = (0, 0, w)$ and the source term as $f(z) = mz$ with constants m, K_z, w the stationary problem reduces to the one-dimensional advection diffusion equation

$$-K_z\, c''(z) + uc'(z) = mz \tag{7.15}$$

which can be solved analytically. It can be easiliy verified that with Dirichlet boundary condition $c(L) = c_L$ at the top of column and a Neumann boundary condition $c'(0) = \zeta$ the solution can be written as

$$c(z) = AK/u\ \exp\{uz/K\} + B + m(Kz + uz^2/2)/u^2 \tag{7.16}$$

where the constants A, B are determined by the boundary conditions via

$$A = \zeta - km/u^2, \quad B = c_L - m(KL + uL^2/2)/u^2 - AK/u \exp\{uL/K\} \tag{7.17}$$

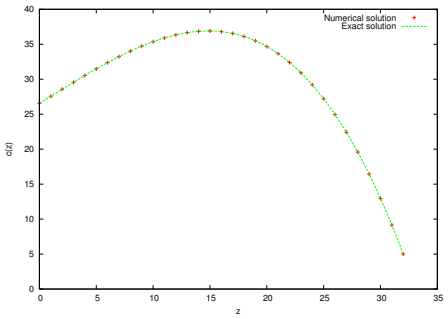

Figure 7.2: Comparison between exact and numerical solution for an diffusion dominated case

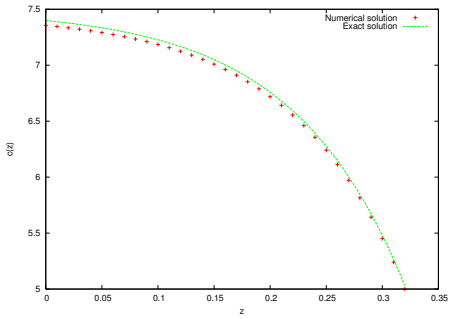

Figure 7.3: Comparison between exact and numerical solution for an advection dominated case

The comparison between the exact solution and the numerical solution for values $K_z = 100, u = 1, m = 1, c_L = 5, \zeta = 1, \Delta z = 1, L = 32$ is given in Fig. 7.2.

The imposition of the Dirichlet condition has been done by setting $\gamma(0) = 10^7$. Note, that different values of K, u, etc might require to adjust $\gamma(0)$ to a higher or lower value in order get a suitably well conditioned linear system of equations.

For an advection dominated case we choose $K_z = 1, u = 10, m = 1, c_L = 5, \zeta = -1, \Delta z = 0.01, L = 32$ and obtain Fig. 7.3 Note, that the solution remains stable (no oscillations) also for quite larger grid spacings Δz whereas the error is drastically increased. It is recommended to apply *a posteriori* error bounds to the advection diffusion problem for the particular cases of interest to better judge the quality of the solution.

7.3 Details of the implementation of parallelism

7.3.1 Background

General

As detailed elsewhere, the snowdrift model is basically an advection diffusion equation for a passive tracer. The old implementation of the model was an explicit Euler scheme for the time integration of the transient advection diffusion equation. Its parallelization was easily accomplished by domain decomposition. Due to the advection dominated behaviour of the equation the scheme was however unstable and an implicit solution was desired. Initiated by Marc Ryser (cf [4]) an semi-implicit scheme (SUPG with Crank Nicolson time stepping) was implemented which requires the solution of a large, unsymmetric, sparse linear system. For an efficient parallel implementation of the solution the whole workflow of the finite element algorithm should be parallelized, i.e. parallel data (vectors, matrices) parallel assembling (domain decomposition) and,

most important for efficiency, a parallel solver (stabilized bi-conjugate gradient)

MPI/PETSc

Since the GRID middleware POPC++ (cf [1]) does not come with parallel libraries for standard numerical problems (linear algebra) we was either forced to re-invent the wheel or to combine well-established libraries with POPC++.

An appropriate starting point is PETSc (cf. [3]) which is an MPI based, parallel-numerics library. Standalone example programs are included in the documentation of PETSc such that a standalone parallel snowdrift model can be implemented in a straightforward manner. Here the main difficulty remains to unify the parallelization in such a manner that the inter-module GRID parallelism achieved by POPC++ (ebalance, snowpack, snowdrift, runoff, etc) can be maintained while introducing additional intra-module parallelism within the snowdrift module with MPI/PETSc.

POPC++ version 1.1

A new (temporary) version of POPC++ was released which regards certain modules as MPI/PETSc processes. In this way, "workers" of a particular modules (such as in snowpack, snowdrift) can be initialized as MPI/PETSc processes and standard MPI/PETSc-code can be simply used in the worker class. Basically, this can be achieved by a new construction mechanism of the SnowDriftWorker class and eventually using the `-''object=petsc''`-flag during linking of the `snowdriftworker.module` In this way the GRID-parallel class `SnowDrift` simply spawns its workers as MPI/PETSc processes. For details I refer to the source code (interface of `SnowDriftParallel` and the construction of the workers in `SnowDriftParallel.cc`)

7.3.2 Implementation details

Node and element enumeration

Ideally the complete parallelization should be done by using solely PETSc which support appropriate data structures with ghost nodes and mapping between different enumeration schemes.

However, here the old structure of the domain decomposition has been maintained, where the whole simulation domain, is cut (by PopC++) into approximately equal chunks along planes parallel to the yz coordinate plane. The implementation always assumes a regular grid where N_x, N_y, N_z are the *global* number of nodes in each coordinate direction. Within such a domain decomposition the (processor-) local numbers of nodes in each coordinate direction, denoted by n_x, n_y, n_z are related to the global values by $N_y = n_y$ and $N_z = n_z$ and only the local n_x is different from N_x. The illustration of the domain decomposition is depicted in Fig. 7.3.2. The overlap (or ghost nodes) between adjacent domains is determined by PopC and contains two layers of nodes. Thus, the global N_x is not simply the sum over the local n_x due to the overlap.

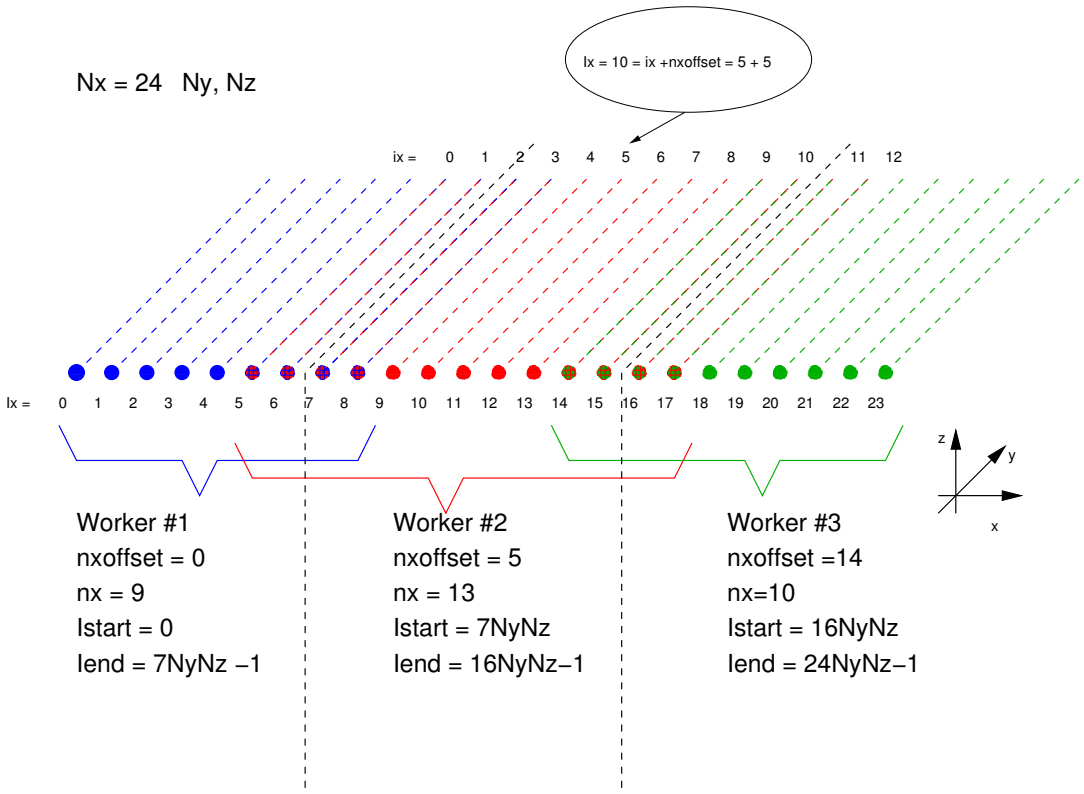

Figure 7.4: Domain decomposition, here for an example with three SnowDriftWorkers and $N_x = 24$. Only the bottom layer of nodes of the whole simulation domain in the xy plane is shown. Some examples values of the variables used in the code are shown.

For the present decomposition it is advantageous to use a node enumeration such that consecutive blocks of node-numbers are hold by one processor. This leads to a global node enumeration as shown in Fig. 7.5: The global mapping of a node with global coordinates I_x, I_y, I_z onto an global node index I is given by $I = I_x N_z N_y + I_z N_y + I_y$, i.e. the lattice is traversed along coordinate directions in the order y, z, x. Global node indices for an element with local element number are available on each processor via the `globalNodeMap` array which is defined in `SnowDriftWorker.cc` It maps the node numbers $\in \{0, 1..8\}$ within an element with local element number $\in \{0, 1..(n_x - 1)(n_y - 1)(n_z - 1) - 1\}$ to the global node index which is e.g. $\in \{5N_yN_z, ..18N_yN_z - 1\}$ for Worker no 2 in the three processor example given above. Note, that the elements are enumerated by traversing the lattice in along coordinate directions in the order x, y, z.

In contrast, locally the nodes are enmuerated also in the x, y, z-scheme which was the old enumeration scheme and which was left for data structure which survived in the present parallel implementation (i.e. the `nodes`-array). More precisely the enumeration is according to $i = i_x + i_y n_x + i_z n_y n_x$ for a node with local coordi-

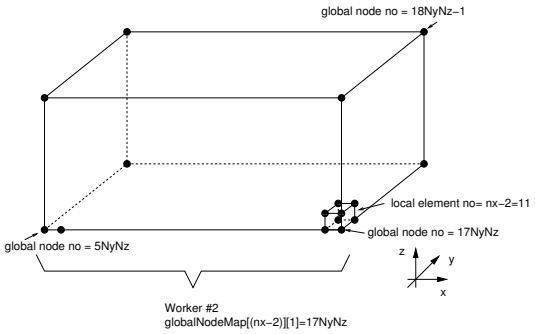

Figure 7.5: Global node enumeration on the example of Worker 2 from the domain decomposition in Fig. 7.3.2

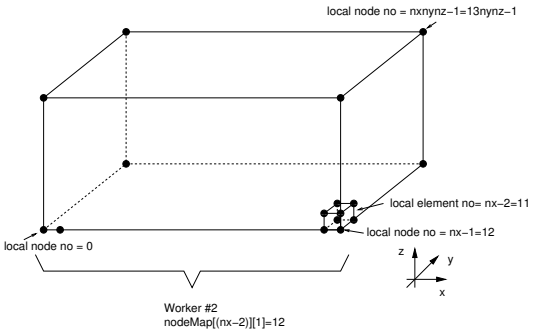

Figure 7.6: Local node enumeration on the example of Worker 2 from the domain decomposition in Fig. 7.3.2

nates i_x, i_y, i_z and local index i. The `nodeMap`-array is defined in `SnowDrift.cc` such that node numbers $\in \{0, 1..8\}$ within an element with a local element number $\in \{0, 1..(n_x - 1)(n_y - 1)(n_z - 1) - 1\}$ is mapped onto the the local node index $\in \{0, 1..n_x n_y n_z - 1\}$. Again, the elements are enumerated in the x, y, z-scheme.

Communication

During the assembling and the solution of the linear system the communication is automatically done by PETSc. The values of the solution vector on the processor domain boundary are afterwards distributed to neighboring processors by POPC++ in the `ExchangeBoundSuspension` method in `SnowDriftWorker.cc`. This is necessary to compute the deposition flux in `SnowDriftFEControl.cc` consistently.

7.3.3 Problems with the present parallel drift

No unifying source code for sequential and parallel mode

The main problem of the present parallel implementation is that it cannot be run sequentially. To explain this difficulty it is necessary to understand the following details of the implementation: Every MPI/PETSc program requires the initialization of the parallel communicator via `PetscInitialize()` which in turn calls `MPI_Init()`. This is implicitely done by POPC++ during construction of the SnowDriftWorkers. This implies i) all PETSc variables (vectors,matrices) have to be members of the `SnowDriftWorker`-class and ii) all workers are automatically ready to execute MPI/PETSc code.

This reveals the two problems when one wants to take over this implementation to a sequential mode: Presently, in sequential mode no workers are constructed. Consequently i) no PETSc variables are available at all ii) and no call to `PetscInitialize()` is done.

A workaround might be possible by (a lot of?) conditional compilation: First, in sequential mode `PetscInitialize()` must be invoked manually somewhere else (where? probably in AlpineMain). Second, in sequential mode all PETSc variables must be members of the class `SnowDrift` and *not* members of the class `SnowDriftWorker`. Vice versa in parallel mode.

Therefore, the parallel version is included in an additional directory `snowdrift_par` in the repository.

Dynamic library completion

It is necessary to additionally hack the snowdriftworker.module after running `make deploy_par` in order to appropriately include the dynamic libraries. This is achieved by the helper application `parocexe` which is located in the `tools` directory located in your `trunk` directory. The call to parocexe is included in the `Makefile.par`.

7.3.4 If you want to use it

1. adjust your environment: `source ./tools/popc-petsc.env`
2. `make all_par -f Makefile.par`
3. `make deploy_par -f Makefile.par`
4. `./sc_drift_par.sh`

With the present "installation" of popc only the MPI processes are computed in parallel, the remaining modules are running on your local machine.

7.4 TODO

1. install a proper version of popc-1.1 in /usr/local against mpi (should be the version `/usr/local/mpich-1.2.5.2` since petsc-2.3.0 is compiled against it) and `petsc-2.3.0-popc`

Chapter 8

Data Assimilation Module

8.1 Files

The data assimilation module in `./assimilation` contains the files

```
DataAssimilation.h
DataAssimilation.ph
DataAssimilation.cc
PackDataAssimilation.cc
```

8.2 Description

The assimilation module (often abbreviated by "DA" or "da" in the code) is responsible for any data assimilation specific computations. Presently, it basically hosts memory for the assimilation data which could have been included also in some of the other modules since only direct insertion like assimilation without any computation is implemented. It has been decided to add an additional module though for the sake of generality since for more sophisticated assimilation schemes (like a Kalman Filter) extensive numerics might be required (i.e. for the computation of the Kalman gain matrix).

The module can be enabled by the switch `-enable-da` in the command line options of Alpine3d limilar to the other modules. Additionally, a path to the assimilation data has to be provided by adding the switch

`-dadir=<path to da-data, without terminating slash>`

Having enabled the data assimilation the basic working steps of Alpine3d which involve the assimilation are given below:

1. At each time step the `Compute()` method of the DataAssimlation class is invoked in `AlpineControl::Run` in `AlpineControl.cc` which forces its input member to try to read assimilation data from the specified directory

dadir. In that directory, the program expects data in files with names <YYYYMMDDHH>.sca which must contain integer data and should be formatted according to the usual Ascii ArcInfo grid format.

2. If data is available in that directory with the correct timestamp it is read into the memory of the assimilation class and send to snowpack, which itself distributes the data to its workers (if run in parallel). If no DA-data is available at that timestep an exception is thrown and everything continues regularly.

3. Within snowpack, Snowpack::Assimlate() is invoked and the DA data is used to control actions in Snowpack. For the present example in the case of SCA data cf Snowpack::Assimilate() in SnowInterface.cc for the details of the actions.

8.3 Modifications in other modules

The modification in other files which were necessary to build in a new class (parclass) are listed below:

- add ./assimlation directory
- modify all Makefiles (for obvious reasons)
- in ./main (creating the DA module in AlpineControl)
- in ./common (add new exceptions, etc)
- in ./snowpack: add pointer to assimilation module, add assimilation routines, add distribution of DA data to workers
- in ./inout: add a method for reading DA-data and a method for getting the grid dimensions (GetGridSize(...))

Bibliography

[1] See documentation on
(`http://gridgroup.tic.hefr.ch/popc/index.php/Main_Page`)

[2] Lehning, M., Völksch, I., Gustafsson, D., Nguyen, T.A., Stähli, M., Zappa, M., (2006) ALPINE3D: A detailed model of mountain surface processes and its application to snow hydrology, *Hydrol. Processes, 20*, 2111-2128.

[3] See documentation on
(`http://www-unix.mcs.anl.gov/petsc/petsc-as/`)

[4] Ryser, M. Numerical Simulation of Snow Transport Above Alpine Terrain Internship Report see `MarcRyser_Drift_Final.pdf`

[5] See documentation on
(`http://www.globus.org/toolkit/`)

[6] Online book on
(`http://svnbook.red-bean.com`)

[7] Jonas, T. (SLF) "Alpine3d Crash Course for the Snow Hydrology Research Group"