Simulations of complex atmospheric flows using GPUs - the model ASAMgpu -

Der Fakultät für Physik und Geowissenschaften

der Universität Leipzig

genehmigte

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Doctor rerum naturalium

Dr. rer. nat

vorgelegt

von	Dipl. Phys. Stefan Horn
geboren am	2. April 1982 in Leipzig
Leipzig, den	8. Oktober 2014
Gutachter	Prof. Dr. Eberhard Renner
	Prof. Dr. Siegfried Raasch
Tag der Verleihung	17. Juli 2015

Bibliographische Beschreibung

Horn, Stefan
Simulations of complex atmospheric flows using GPUs
- the model ASAMgpu -
Universität Leipzig, Dissertation
106 S.[1], 33 Zitate[2], 40 Abbildungen, 24 Programmausdrucke

Referat:
Die vorliegende Arbeit beschreibt die Entwicklung des hochauflösenden Atmosphärenmodells ASAMgpu. Dabei handelt es sich um ein sogenanntes Grobstrukturmodell bei dem gröbere Strukturen mit typischen Skalen von Deka- bis Kilometern in der atmosphärischen Grenzschicht explizit aufgelöst werden. Hochfrequentere Anteile und deren Dissipation müssen dabei entweder explizit mit einem Turbulenzmodell oder, wie im Falle des beschriebenen Modells, implizit behandelt werden. Dazu wurde der Advektionsoperator mit einem dissipativen Upwind-Verfahren dritter Ordnung diskretisiert. Das Modell beinhaltet ein Zwei-Momenten-Schema zur Beschreibung mikrophysikalischer Prozesse. Ein weiterer wichtiger Aspekt ist die verwendete thermodynamische Variable, die einige Vorteile herkömmlicher Ansätze vereint. Im Falle adiabatischer Prozesse stellt sie eine Erhaltungsgröße dar und die Quellen und Senken im Falle von Phasenumwandlungen sind leicht ableitbar. Außerdem können die benötigten Größen Temperatur und Druck explizit berechnet werden. Das gesamte Modell wurde in C++ implementiert und verwendet OpenGL und die OpenGL Shader Language (GLSL) um die nötigen Berechnungen auf Grafikkarten durchzuführen. Durch diesen Ansatz können genannte Simulationen, für die bisher Supercomputer nötig waren, sehr preisgünstig und energieeffizient durchgeführt werden. Neben der Modellbeschreibung werden die Ergebnisse einiger erfolgreicher Test-Simulationen, darunter drei Fälle mit mariner bewölkter Grenzschicht mit flacher Cumulusbewölkung, vorgestellt.

[1] S. (Seitenzahl insgesamt)
[2] Anzahl der im Literaturverzeichnis ausgewiesenen Literaturangaben

Contents

1 Introduction

One of the oldest and very successful applications of numerical models in the field of physics is the simulation and prediction of atmospheric processes. Already in the early decades of the last century, shortly after Abbe recognized that the atmosphere can be described by fundamental hydro- and thermodynamics (Abbe [1910]) and Richardson applied numerical methods to perform the first physical based weather model forecast in history, the idea of parallelizing this process emerged (Richardson [1922]). Richardson realized that the enormous number of calculations needed to produce any kind of global weather forecast could never be accomplished by a single person. The calculation process had to be distributed over a large number of workers controlled by some kind of organizer. This leads to the idea of the Richardson's Forecast Factory. In his book from 1922 on page 219 he wrote:

"Imagine a large hall like a theatre, except that the circles and galleries go right round through the space usually occupied by the stage. The walls of this chamber are painted to form a map of the globe. The ceiling represents the north polar regions, England is in the gallery, the tropics in the upper circle, Australia on the dress circle and the antarctic in the pit. A myriad computers are at work upon the weather of the part of the map where each sits, but each computer attends only to one equation or part of an equation. The work of each region is coordinated by an official of higher rank. Numerous little "night signs" display the instantaneous values so that neighbouring computers can read them. Each number is thus displayed in three adjacent zones so as to maintain communication to the North and South on the map. From the floor of the pit a tall pillar rises to half the height of the hall. It carries a large pulpit on its top. In this sits the man in charge of the whole theatre; he is surrounded by several assistants and messengers. One of his duties is to maintain a uniform speed of progress in all parts of the globe. In this respect he is like the conductor of an orchestra in which the instruments are slide-rules and calculating machines."

It is amazing how well this passage describes basic architectural features of modern high performance computer clusters and accordingly the structure

of the computational cores in modern graphics adapters. While actual central processing units (CPUs) consist of ten to twenty computational cores with very high clock speeds and a huge set of available operations, the processing units in graphics adapters (GPUs) feature several hundreds up to thousands of such workers with a reduced instructional set and slightly lower clock cycle. The question arises: "Is it possible and what is necessary to utilize this computational power for weather forecasting." To answer this question this work gives an introduction to one possible way to use GPUs for general purpose computations.

Therefore two classes were developed to abstract the GPU access functions provided by OpenGL into a simple buffer kernel framework. The second section of this work will give a detailed introduction to those classes and an example how to apply this framework to a simple shallow water equation system.

The classes were then used to implement a three dimensional moist atmospheric model (ASAMgpu) using an explicit time integration and a two moment microphysical scheme. Currently operational high resolution forecast weather models like the COSMO model (Steppeler et al. [2003]) by the German Weather Service or the Weather Research and Forcasting Model WRF (Skamarock and Klemp [2008]) by the NOAA usually use cell sizes in the range from 2 to 4 km. With increasing computational power this resolution will increase enabeling processes like boundary layer clouds and turbulent mixing processes to be explicitly resolved. With that, more complex approaches for microphysical parameterizations can be included in weather prediction applications. The new model ASAMgpu focuses on domain sizes and grid cell dimensions commonly used in large eddy simluation models (LES). Those models resolve larger turbulent structures explicitly and have to handle sub grid scale turbulence and viscosity using explicit parameterizations or implicit numerical approaches (ILES, Hickel [2008]). Typical grid cell sizes for boundary layer convection range from ten to a few hundred meters, for processes at cloud boundaries even smaller cells in dependence on the studied process are used. There are already several existing LES models for example PALM (Raasch and Schröter [2001]), DALES (Heus et al. [2010])

or the UCLA-LES (Stevens et al. [2005]) model, just to name a few. Also some regional models like the WRF model can be used in a LES mode as well (Moeng et al. [2007]). Most of them use clusters of common CPUs connected by some kind of network infrastructure and a message passing interface for the communication between the nodes. Compared to GPUs the number of grid cells handled by one single node is small, which produces high communication costs. The process of miniaturisation of computational devices and minimization of communication leads directly to a cluster infrastructure of CPU nodes utilizing one or more GPUs each. This idea was also followed by the work of Michalakes and Vachharajani [2008], where one possible microphysical parameterisation available in the WRF model was transferred to be processed on a GPU. This brought a twenty times increase in performance for the microphysics alone and overall 1.3 times increase for the total application. One factor limiting the speedup was the communication cost between the CPU memory space and the GPU memory, because large three dimensional fields had to be transferred through the relatively slow PCIe bus. To minimize communication cost through that channel and increase overall speedup it makes sense to perform as much calculations as possible on the GPU, including the dynamics, and to transfer data only for the boundaries between the GPU nodes and if necessary the CPU nodes respectively. This approach was also followed during the implementation of GALES by Schalkwijk et al. [2012], which is a completely rewritten version of the DALES code using C++ and modern CUDA utilities done at the TU Delft. The model ASAMgpu also follows this approach to perform all calculations on the GPU and minimize communication, even some of the analysis and plotting during runtime is performed on the GPU to prevent saving large datafields during every timestep. This enables visualisation of data at very high temporal resolution. That direct feedback during a model run simplifies identifying problems, like high frequency oscillations for example. During the development of the ASAMgpu model a new thermodynamic variable was used, which for moist atmospheric processes has some advantages over previously used variables like the internal energy, entropy or the equivalent potential temperature for example. The used variable is a form of the entropy, and with that

is conserved under adiabatic processes, needed physical quantities like pressure or temperature can be calculated explicitly and the source terms during phase transitions can be derived very easily. Model details like the governing equations, the used time integration scheme, the used thermodynamic variable and the implemented microphysical parameterisations are presented in the third section of the present work.

In the fourth section several test applications of the model ASAMgpu, using the introduced framework, are presented. The first test is a simulation of a Rayleigh-Bénard convection (Manneville [2006]) between a heated plate at the lower surface of the domain and a cooled plate at the top. This was used to check the model for breaks in symmetry due to floating point mistakes and to get an idea of the behaviour of such a simple boundary layer convection system without explicit turbulence parameterisation. The second and third example are a rising heat bubble in a dry adiabatic layered atmosphere by (Wicker and Skamarock [1998]) and a dry cold bubble falling on the ground developing a strong density current (Straka et al. [1993]). Both tests mainly focus on the stability of the used time integration mechanism. The fourth very idealized test case was a moist heat bubble in a slightly supersaturated environment (Bryan and Fritsch [2002]). This test was performed to validate the implemented condensation mechanism and the amount of released latent heat. The theoretical solution is similar to the dry heat bubble test case, and also the results of the numerical simulation fits quite well.

The next step was to use more realistic setups to simulate boundary turbulence in combination with cloud processes at the top of the boundary layer. To be able to compare the results of the simulations with results from similar models, three cases from the *Global Energy and Water Cycle Experiment Cloud System Studies* (GCSS) were choosen (GEWEX Cloud System Science Team [1993]). All the three cases describe marine cloud topped boundary layer systems, with relatively shallow boundary layers, an inversion and clouds developing below that inversion. The first one is the *Barbados Oceanographic and Meteorological Experiment* (BOMEX, Siebesma et al. [2003]) case, which describes a non drizzling marine cumulus cloud layer. The vertical profiles and time series of the simulation results are presented in section

4.5. The second GCSS case is from the *Second Dynamics and Chemistry of Marine Stratocumulus field study* (DYCOMS-II, Stevens et al. [2003]). The case describes a nocturnal stratocumulus cloud deck under a strong inversion. The third case is the *Rain in Cumulus over the Ocean* (RICO) case (VanZanten et al. [2011]), based on the campaign (Rauber et al. [2007]) that took place during November 2004 – January 2005 above the western atlantic. This case features marine cumulus clouds including precipitation.

Finally one interesting pratical application was the simulation of the influence of an heated island surface on the marine boundary layer. During the *Second Saharan Mineral Dust Experiment* (SAMUM2, Ansmann et al. [2011]) campaign, Doppler lidar measurements on the island Santiago, which is the largest of the Cap Verde islands, were performed. One objective of this measurement campaign was the investigation of aerosol entrainment from elevated aerosol layers down into the marine boundary layer. With that the representativity of on-shore lidar measurements for the marine boundary layer was of high interest. Also the origin of some reccuring patterns in the vertical velocity measurements were not clear. The performed simulations helped during the interpretation of these results.

This work is closed by a summary, a short overview over some realized alternative applications of the presented buffer-kernel-framework and an outlook for possible future development of the ASAMgpu model.

2 General purpose computation on graphics processing units

The first part of this section gives an overview about general purpose computation on graphics processing units, and it explains what OpenGL and GLSL are and why they were chosen. The following subsections give a more detailed introduction into GPU programming, the developed shader and texture classes and the used frameworks, including code examples which should enable the reader to follow the development. It is finalized by the application of the developed classes for a simple shallow water example.

2.1 OpenGL + GLSL

The model ASAMgpu uses modern graphics processing units (GPUs) for fast and efficient massive parallelized computations. The application of GPUs for non-graphic calculations is also called general purpose computation on graphics processing units (GPGPU). During the last years many different approaches for GPGPU were developed. One early way to program GPU hardware was the Brook+ Library by ATI. Later more and more vendor specific solutions like CUDA from nVidia or Stream from ATI evolved. One disadvantage of these solutions was the hardware specific implementation and with that the platform and vendor dependence of the software. The decision for OpenGL was mainly inspired by personal experiences from the years when the first consumer graphics cards with support of hardware accelerated three dimensional graphics appeared on the market at the end of the last millenium. The driver support for those graphics devices was as fragmented as the GPGPU market is today. For example the GLIDE drivers for the Voodoo Graphics cards by 3dfx Interactive, a company which was bought by nVidia later, provided a slight performance boost for 3D gaming applications compared to the early Direct3D versions from Microsoft and the already existing OpenGL standard. At this time OpenGL was mainly used by scientific and CAD software. 3dfx followed an aggressive promotion strategy to place emphasis in the gaming market to their GLIDE API

and with that to introduce a vendor lock into the published games to force the costumers to buy 3dfx hardware. It took a while until they were finally forced in 1996 by the game "Quake" by Id Software to provide a minimal working OpenGL driver. While the focus of OpenGL and DirectX was to hide the technical details behind a software layer and in case of OpenGL also to provide a platform independent API, the GLIDE drivers allowed low level hardware access and with that an increased performance compared to the other solutions. With time, more sophisticated device drivers and their growing support for the open and independent APIs, outweighted the small perfomance boost and GLIDE became obsolete. Similar tendencies and aggressive marketing strategies can be observed concerning nVidias CUDA and their TESLA platforms in todays GPGPU market, including the risk that CUDA and software depending on CUDA will become obsolete as well.

In contrast to CUDA, OpenGL was developed by SGI and published in 1992 as an open and cross platform programming interface to access GPUs and it is still under development. It is currently specified and developed by the Khronos Group, which is a non profit consortium with members like 3Dlabs, ATI/AMD, Discreet, Intel, nVidia, SGI, Sun Microsystems, Google, Apple, Samsung and many more. The model ASAMgpu uses the OpenGL approach with the OpenGL Shader Language (GLSL). The Khronos group also developed the Open Computing Language (OpenCL) as an interface to GPU and CPU parallelization. Because at the time the development of ASAMgpu started, OpenCL was still a theoretical construct and OpenGL was available for a long time as a very reliable API with support for a wide variety of hardware, OpenGL was used. A future transition from OpenGL to OpenCL should be straight forward because the overall code structure is compatible.

The main concepts of all GPGPU solutions are quite similar, but the vocabulary is different between the different approaches. The main idea is the so called stream processing paradigm. That means, the computation process is defined using a certain number of streams, also called buffers or textures. Those streams are used to access the data describing the physical system in the memory of the graphics device. Small programs, called

kernels or shaders, define the computations applied to those buffers. In the used OpenGL approach these kernels are written in GLSL, a language with a C-like syntax. GLSL supports functions, if-branches and loops. Branches should be used carefully because they could significantly slow down the computation step. After the definition of the needed input and output buffers and the used kernel, a calculation step is initiated. Using OpenGL this is realized by drawing a rectangle using a certain number of input textures and redirect the output from the screen to one or more output textures. The number of simultaneous possible input and output textures differs from device to device, currently eight for each is a common number. During this step, the graphics device splits the output buffers into a huge number of blocks, which then are all processed in parallel by the shader units, also called stream processors. In OpenGL this is done by the device driver completely transparent to the developer. In OpenCL and vendor specific solutions like CUDA, it is possible to control block sizes and the number of used threads. Current GPUs have a large number of shader units, e.g. an ATI Radeon HD 5870 (2009) has 320 independent stream processors where every single one is a small 5D vector unit, resulting in processing up to 1600 buffer elements in one clock cycle, nVidia Tesla C2070 (2011) with 448 scalar stream processors at higher clock cycles compared to the HD 5870. More recent examples are the AMD R9 290X (2013) with 2816 and Tesla K40 (2013) with 2880 stream processors. Newer cards not only provide more computation cores but lower power consumption and larger graphics memory as well.

2.1.1 OpenGL context

Since GPUs were build for graphic calculations, the intuitive way to access this hardware is through the graphics API of the used operating system. In a Linux environment this is in most cases the xserver using the appropriate device driver which provides a hardware accelerated graphics context. Therefore some kind of access to the desktop manager (e.g. Gnome, KDE, Unity) is needed to open a window including the graphics context. In our case a very reliable and well documented way is to use the Simple DirectMedia Layer

(http://www.libsdl.org/). This library provides a lot of functions mainly focused on game development and is available for a wide variety of operating systems. The few SDL function calls needed are implemented in the constructor of a new class with the name *openglcontext* (header file shown in listing 1, implementation source code in listing 19 in the appendix).

```
1 class openglcontext
2 {
3  public:
4   int sizex, sizey;
5
6   openglcontext(int sizexn, int sizeyn);
7 };
```

Listing 1 The header file for the *openglcontext* class. (openglcontext.h)

This class is used to open the needed calculation window, which allows output and direct user interaction as well, with just one single C++ instruction. The *openglcontext* constructor initialises the SDL library, opens a window providing the hardware accelerated drawing context, checks for some necessary OpenGL features like using custom fragment shaders and sets some needed OpenGL settings. After the construction of such an *openglcontext* object the application may use standard OpenGL commands for GPU access. An example for the usage of the constructor is shown in Listing 2. The parameters passed to the constructor define the size of the resulting window, in this case a width of 800 pixels and a height of 600 pixels.

```
1 #include "openglcontext.h"
2
3 int main ( )
4 {
5  openglcontext context(800,600);
6 }
```

Listing 2 This main source demonstrates the usage of the *openglcontext* class to open a window containing an OpenGL context for computation/drawing.

This simple example and the *openglcontext* class can be compiled using the following commands.

```
> g++ -c openglcontext.cpp
> g++ main01.cpp openglcontext.o -lSDL -lGL -lGLU -lGLEW
> ./a.out
```

The four linker flags in the second command ensure that the SDL library, the OpenGL (GL) library, the OpenGL Utility (GLU) library and the OpenGL extension Wrapper (GLEW) library are linked to the application. The so produced executable file *./a.out* opens up a window and, while nothing else is written in the main function, closes it immediately.

2.1.2 Shader/Kernel

The term shader derives from the original task these small programs had, which was to calculate the resulting color of a projected pixel from the surface of a 3D object using certain lighting and shading models in computer graphics, but with the invention of the unified shader architecture they now can be used for a variety of other things as well. OpenGL distinguishes different types of shader programs by their task during the render process of a three dimensional scene. In the current render pipeline those are the vertex shader, the geometry shader, the tesselation shader and the fragment shader. The vertex shader is applied to every point describing the geometry, and is mainly used to apply the projection matrix to those vertices to calculate the final position on the two dimensional projection plane in the resulting image. Here additional effects on the geometry may be applied, for example wave like deformations to generate an ocean like surface. The geometry shader can be used to generate complex geometrical shapes from a single vertex information as the input parameter. This is useful for rendering large numbers of more or less similar objects like grass, trees in a forest or a large army in a computer game. The tesselation shader was introduced with OpenGL V4.0 and can be used to subdivide surface patches to generate very smooth geometries even for close viewing distances. The task of the fragment shader, sometimes also called the pixel shader, is to fill the area between the resulting vertices on the projection plane and to calculate the resulting color for every pixel between those vertices. Therefore every pixel

is processed independently and in parallel to all other pixels. In general, shaders are able to use texture information as input fields. This is the step, where the GPGPU calculations can be performed. Each pixel in the texture represents an element of the system, like a gridcell or a particle for example. The different color channels for that pixel can be interpreted as physical properties of the according element. In every render step these properties can be changed in dependence of the parameters of all the other elements saved in the input texture. The output result is then written into a new texture and can be used as input data in the next iteration step. With that it is quite simple to realize an explicit time integration scheme. Because all those element updates should be independent from each other, they can easily be parallelized by the device driver and distributed among the available shader devices. But at first in Listing 3 the *shader* class definition in the shader header file is presented.

```
1 #include <GL/glew.h>
2 #include "texture.h"
3
4 class shader
5 {
6  public:
7   GLhandleARB Handle, VertexHandle, FragmentHandle;
8   GLint numTexUnits, numDrawBuffers;
9   GLuint fb;
10  texture **t,**tout;
11
12  shader(const char *header, const char *filename);
13
14  void bind();
15  void step();
16 };
17
18 #endif
```

Listing 3 This is the header file for the *shader* class, which is used to load GLSL shaders from an ASCII file and bind it into the render pipeline to perform calculations.

The *shader* class basically implements three methods. The first function is the constructor for any *shader* object, including loading the shader source code from an ASCII text file, transfer the code to the device driver, compile the source code and return a handle which is used to refer to the shader for further usage. The second function is called *bind* and is used to assign input and output textures to the corresponding texture devices which then can be used during shader execution. Therefore the *shader* object contains two fields of pointers to texture objects (***t,**tout*) which have to be set accordingly before calling the *bind* method. This will be explained in detail later in this section. The third function (*step*) is the actual render step, where a rectangle is drawn with the size of the target texture. During this step the GPU actually performs the calculations. The following basic example demonstrates the usage of the *shader* class and the structure of a simple vertex and fragment shader. While only the fragment shader is used for GPGPU calculations later, the vertex shader is still essential. The example shows how to load and bind the shader into the render pipeline. Hence the *texture* class has not been introduced yet, this example does not use any textures as input or output buffers and the result is directly written into the framebuffer and displayed on the screen. This example fragment shader just returns the color red.

```
1  #include <SDL/SDL.h>
2  #include "openglcontext.h"
3  #include "shader.h"
4
5  int main()
6  {
7   openglcontext context(800,600);
8   shader *s;
9   s=new shader(NULL,"shaders/red");
10
11  for(int i=0;i<100;i++)
12  {
13   s->bind();
14   s->step();
15   SDL_GL_SwapBuffers();
```

```
16   }
17 }
```

Listing 4 This main source code demonstrates the basic usage of the *shader* class.

As shown in listing 4 the new shader object is constructed in line 9 using the shader source code saved in a textfile with the relative path "shaders/red". Therefore two more essential files have to exist in the shader subdirectory. The files *red.fsd* and *red.vsd* contain the source code for the fragment shader (.fsd) and the vertex shader (.vsd). For each a basic example is given in listings 5 and 6. Then the example just binds and executes the shader a thousand times, after every shader execution a *SDL_GL_SwapBuffers* has to be called, to get the result from the back framebuffer to the front framebuffer. This is a common technique for animation, where during the calculation the resulting image is drawn in a hidden backbuffer while the actual displayed framebuffer is held constant. When the render process has finished the two buffers are flipped and the display gets an updated image as fast as possible to prevent flickering.

```
1 void main()
2 {
3   gl_Position    =gl_ModelViewProjectionMatrix*vec4(gl_Vertex);
4 }
```

Listing 5 The vertex shader source code, which is used to calulate the position in the output buffer of the resulting pixel.

During the *step* method of the *shader* class a rectangle is drawn with the same edge points as the OpenGL context window. The given vertices are processed by the vertex shader. To get the final position of the resulting fragment on the screen, or alternatively the target texture, the incoming vertex is multiplied with the actual projection matrix, which is in our case a simple orthographic projection viewed from the top. After the vertex shader execution the fragment shader is used to calculate the resulting color for every pixel in the requested fragment (in this case the whole output area in the window or the texture). This is the step where the fine grain parallelization of the GPU is used. That means, the large number of resulting pixels, the target

fragment contains, is distributed by the device driver among the available shader devices and those perform the calculation simultaneously. In the following simple example the fragment shader returns the color red for all pixels in the fragment.

```
1 void main()
2 {
3  gl_FragColor=vec4(1.0,0.0,0.0,1.0);
4 }
```

Listing 6 The fragment shader source code is used to compute the resulting color for the processed pixel.

The code can be compiled and executed using the following commands. The *shader* class depends on the *texture* class, so this one has to be build and linked as well, although it is not explicitly used in this example. In addition the compiler call has to be extended to link the *libgd2* library which is used to load and write images in the *texture* class. More details can be found in the next section.

```
> g++ -c openglcontext.cpp
> g++ -c texture.cpp
> g++ -c shader.cpp
> g++ main02.cpp texture.o shader.o openglcontext.o -lSDL \
      -lGL -lGLU -lGLEW -lgd
> ./a.out
```

If the device driver was installed properly, the program opens a window and fills it with the color red. The next step is to send some data from the CPU to the GPU and produce some effect in the fragment shader. Therefore OpenGL provides a set of functions which can be used to send different types and amount of data to the actually bounded shader. In this case the function *glUniform1iARB* (line 14 in listing 7) is used to send one integer variable (i) to the shader. This function has to be called between the *bind* and the *step* method. The keyword *uniform* during the variable definition in the fragment shader (listing 8) allows access to the incoming value and can be used to parameterize the result. For example the amount of the green

channel in the returned color could be varied to produce a blinking effect. This technique can be used to transfer a few parameters from the CPU to the GPU. For large datafields the texture class, introduced in the next section should be used, because there data transfers will be asynchronous and in parallel.

```
#include <SDL/SDL.h>
#include "openglcontext.h"
#include "shader.h"

int main()
{
 openglcontext context(800,600);
 shader *s;
 s=new shader(NULL,"shaders/blink");

 for(int i=0;i<100;i++)
 {
  s->bind();
  glUniform1iARB(glGetUniformLocationARB(s->Handle,"i"),i);
  s->step();
  SDL_GL_SwapBuffers();
 }
}
```

Listing 7 Basic example for how to transfer a single control parameter to a shader program. In this example one integer (i) for the iteration step is transferred.

```
uniform int i;

void main()
{
 float t=float(i)/60.;
 gl_FragColor=vec4(1.0,0.5+0.5*sin(20.*t),0.0,1.0);
}
```

Listing 8 Basic example for usage of the transferred parameter in a fragment shader.

During evaluation of more complex systems the position of the currently processed grid cell, and if necessary their neighbors, has to be known. This

position is calculated in the vertex shader and can be passed using the keyword *varying* during variable definition in the vertex and fragment shader. With that the current grid cell can be identified and a huge variety of procedural textures can be calculated. Listings 9, 10 and 11 show an example for passing several control parameters to a fragment shader that uses the position to calculate a two dimensional procedural texture.

```
1  #include <SDL/SDL.h>
2  #include <stdlib.h>
3  #include "openglcontext.h"
4  #include "texture.h"
5  #include "shader.h"
6
7  int main()
8  {
9   openglcontext context(800,600);
10  shader *s;
11  texture *t;
12
13  s=new shader(NULL,"shaders/procedural");
14  t=new texture(800,600,1);
15
16  float w[32];
17  float v[32];
18  srand(100);
19  for(int i=0;i<32;i++)
20   w[i]= 20.+40.*float(rand())/float(RAND_MAX);
21  for(int i=0;i<32;i++)
22   v[i]=-20.+40.*float(rand())/float(RAND_MAX);
23
24  for(int i=0;i<500;i++)
25  {
26   s->bind();
27   glUniform1iARB(glGetUniformLocationARB(s->Handle,"i"),i);
28   glUniform1fvARB(glGetUniformLocationARB(s->Handle,"w"),10,w);
29   glUniform1fvARB(glGetUniformLocationARB(s->Handle,"v"),10,v);
30   s->step();
31   SDL_GL_SwapBuffers();
32  }
```

```
33 }
```

Listing 9 This is the main source file (main.cpp) for an example that produces an animated procedural texture and shows it on the screen. In this example 10 random floats in the arrays w and v, and one integer for the iteration step are transferred.

```
1 varying vec2 pos;
2
3 void main()
4 {
5  gl_Position=gl_ModelViewProjectionMatrix*vec4(gl_Vertex);
6  pos         =gl_Position.xy;
7 }
```

Listing 10 The vertex shader source code for the animated procedural texture example.

```
1 uniform int i;
2 uniform float w[32];
3 uniform float v[32];
4
5 varying vec2 pos;
6
7 void main()
8 {
9  int n=32;
10  float t=float(i)/60.;
11
12  float r,g,b;
13  r=0.0;g=0.0;b=0.0;
14
15  for(int j=0;j<n;j++)
16  {
17   r+=(0.5+0.5*sin(pos.x*w[j]+v[j]*t));
18   g+=(0.5+0.5*sin(pos.y*w[j]+v[j]*t));
19   b+=(0.0+1.0*sin(20.*length(pos)+w[j]*t+w[j]));
20  }
21
22  gl_FragColor=vec4(r/float(n),g/float(n),b/float(n),1.0);
23 }
```

Listing 11 The fragment shader source code for the animated procedural texture example. The resulting color is a function of the output position and the iteration (i).

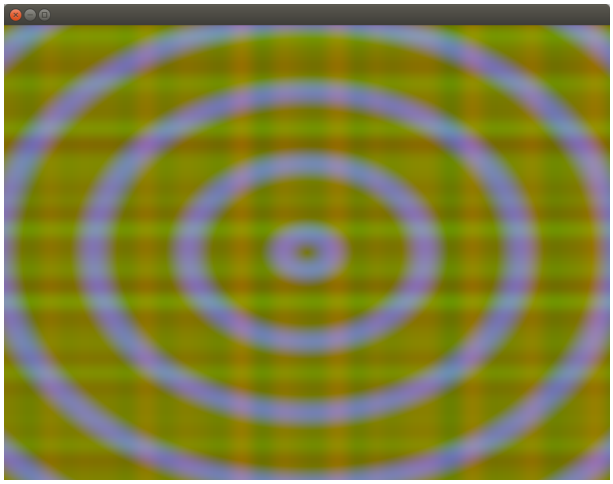

Figure 1 Snapshot of the output of the procedural texture example. The displayed lines and circles are calculated in dependence of the parameters, passed from the main source to the fragment shader.

GLSL may be very strict with typing on some devices, depending on the device driver. So with some drivers a floating point variable only accepts floating point values, no implicit casting is allowed. One possible result of the procedural texture example is shown in figure 1. In all examples in this section the results are written into the framebuffer to display them on the screen. After the render process has finished all results are lost. To get access to the results the target memory address for the render process has to be changed. With that the output is redirected into a buffer in the GPU memory, also referred to as a texture, and is then available to further render steps. This procedure will be explained in the following section.

2.1.3 Textures/Buffers

The examples, presented in the last chapter, only used one single render pass to explicitly generate larger data fields and displayed them directly on the screen. In this chapter we will at first use an existing data field, upload it to the GPU and use it as an input field for an example shader to display the data on the screen. The second part of this section will focus on the redirection of the output data into another texture to use it in the next render step. With

that a simple iteration process can be realized, for example for an explicit time integration scheme. This will be demonstrated using a simple shallow water equation system.

As an introduction, the header file of the *texture* class:

```
1 #include <GL/gl.h>
2 #include <stdio.h>
3
4 class texture
5 {
6  public:
7   GLuint id,fb;
8   int sx,sy,sz;
9   float *data;
10
11   texture(int sxn,int syn,int szn);
12   int getindex(int x,int y,int z);
13   void ram2gpu();
14   void gpu2ram();
15   void set(int x,int y,int z,int c,float v);
16   float get(int x,int y,int z,int c);
17   void save(const char *filename);
18   int load(const char *filename);
19   void loadpng(const char *filename);
20   void savepng(const char *filename);
21   void savejpg(const char *filename);
22 };
```

Listing 12 The header file for the *texture* class.

Because GPUs are optimized for two dimensional texture access with up to four components per pixel, the *texture* class used in this work, maps a three dimensional field with four floating point values per cell onto a two dimensional OpenGL texture field. That is why the constructor for a texture object needs three parameters, the domain size in the x, y and z dimension respectively, but constructs a two dimensional OpenGL texture object. The *data* pointer points to the first element of the texture data in CPU memory. The *getindex* method is used to transfer the coordinate from three dimensional model space into the one dimensional address (the index) in the *data*

field.

The two methods *ram2gpu* and *gpu2ram* transfer the data field from CPU- to GPU memory and vice versa. So *gpu2ram* can be used to retrieve data from a GPU texture buffer and copy it to CPU memory to finally save it to disk or perform further analysis. Compared to CPU memory access or GPU onboard memory access, this data transfer is quite slow and should be done only if really necessary.

The *set* and *get* methods are useful for initializing texture fields in CPU memory. After the usage of the *set* method, *ram2gpu* has to be called to copy the updated values to the GPU memory. *Save* and *load* are used to save data from CPU memory into a raw binary file or load it from there. For loading texture data from a file, the texture size has to fit to the input file. The method *loadpng* may be used to load an image and use it as an input data field. Last but not least, there are the two methods *savepng* and *savejpg* to save the data field using a compressed image format into a file, to directly show it in a viewer or use it for further visualization. Those two methods and the *loadpng* method use function calls of the common gd2-library (http://www.boutell.com/gd/). Hence the image file access functions only support eight bits per channel, they are not supposed to be used for data saving and loading, but for visualisation purposes only.

The following code block (listing 13) shows the first example for the usage of the *texture* class, where a *texture* object is constructed, data is loaded from an existing example image file and used as an input field for a simple shader that just passes the texture information to the framebuffer to show the image in the open window.

```
1 #include <SDL/SDL.h>
2 #include <stdlib.h>
3 #include "openglcontext.h"
4 #include "shader.h"
5 #include "texture.h"
6
7 int main()
8 {
9  openglcontext context(800,600);
```

```
10  shader *s;
11  texture *t;
12
13  s=new shader(NULL,"shaders/texture");
14  t=new texture(800,600,1);
15  t->loadpng("test.png");
16  t->ram2gpu();
17
18  for(int i=0;i<500;i++)
19  {
20   s->t[0]=t;
21   s->bind();
22   s->step();
23   SDL_GL_SwapBuffers();
24  }
25 }
```

Listing 13 Main source code for the usage of the *texture* function. The program loads an example texture from a file and displays it on the screen.

The created *texture* object has a vertical (z) size of one, resulting in a two dimensional image. Using the method *loadpng* a prepared imagefile is loaded into the data field. After that the data is copied to the GPU using *ram2gpu*. To access the data in a shader kernel, the texture has to be bound to a texture device. This is done in line 20 where the *t[0]* pointer is set to the created texture object. The *bind* method will then take account on binding the texture to the appropriate device (in this case the first available texture device) and passes the texture handle to the shader. Listings 14 and 15 show how the texture data has to be accessed in the shader codes. Therefore the vertex shader has to be extended to pass texture coordinates to the fragment shader (line 4), where those are accessible through the built in array *gl_TexCoords*. Texture coordinates are defined in the shader class during the *step* method. In ASAMgpu they are two dimensional and range from zero to one. So $(0, 0)$ refers to the lower left corner of the texture and $(1, 1)$ to the upper right. The built-in variable *gl MultiTexCoord0* is interpolated by the GPU texture device to match to the current output position. The function *texture2D* in the fragment shader then returns the

four dimensional data from the texture for the given position.

```
1 void main()
2 {
3  gl_Position    =gl_ModelViewProjectionMatrix*vec4(gl_Vertex);
4  gl_TexCoord[0]=gl_MultiTexCoord0;
5 }
```

Listing 14 The vertex shader for the *texture* usage example. The so called texture coordinate has to be passed to the fragment shader and contains the position of the resulting pixel.

```
1 uniform sampler2D t0;
2
3 void main()
4 {
5  gl_FragData[0]=texture2D(t0, vec2(gl_TexCoord[0]));
6 }
```

Listing 15 The fragment shader for the *texture* usage example. The texture coordinate is used to retrieve texture information using the built in *texture2D* function.

The result of this texture loading and drawing procedures is shown in figure 2.

Figure 2 Result of a simple example that loads a texture from a file and displays it on the screen.

The second example in this chapter redirects the render output into a texture. This is done by setting the *tout* pointer of the *shader* class to the desired target texture. Hence OpenGL is able to use several render targets, *tout* is implemented as an array. The maximum number of available render targets is hardware dependent, currently eight is a common number. It is not recommended to use an input texture as an output texture during the same render pass, because that will produce so called racing conditions during parallelization and may lead to unexpected results. The model ASAMgpu uses up to four render targets simultaneously. The *shader* class will then organize the creation of drawbuffers for the target textures and connects the output of the shader to the target textures drawbuffers. If the *tout* pointer is set to *NULL* the output of the shader will be directed to the display framebuffer.

The build in array *gl_FragData* provides write access to the available output framebuffers. For example in listing 15 only the first output buffer (with the index zero) is used for writing. The main source code is shown in listing 16. The complete program simply loads the file *test.png*, copies the data from texture *t1* to texture *t2*, and writes it back to the file *save.png*.

```
1 #include <SDL/SDL.h>
2 #include <stdlib.h>
3 #include "openglcontext.h"
4 #include "shader.h"
5 #include "texture.h"
6
7 int main()
8 {
9  openglcontext context(800,600);
10  shader *s;
11  texture *t1,*t2;
12
13  s =new shader(NULL,"shaders/texture");
14  t1=new texture(800,600,1);
15  t2=new texture(800,600,1);
16  t->loadpng("test.png");
17  t->ram2gpu();
18
```

```
19   s->t[0]=t1;
20   s->tout[0]=t2;
21   s->bind();
22   s->step();
23
24   t2->gpu2ram();
25   t2->savepng("save.png");
26 }
```

Listing 16 Main source code for using a texture as an output buffer and save the result to the disk.

Iterative algorithms can be realized by introducing a loop where the output of the shader is used as an input field for the next render step . The main changes to the former example are the commands for swapping the texture pointers (listing 17, line 26-28). With that the output texture becomes the input texture for the next step and vice versa. Explicit time integration mechanisms can be implemented by using this structure.

```
1 #include <SDL/SDL.h>
2 #include <stdlib.h>
3 #include "openglcontext.h"
4 #include "shader.h"
5 #include "texture.h"
6
7 int main()
8 {
9   openglcontext context(800,600);
10  shader *s;
11  texture *t1,*t2,*temp;
12
13  s =new shader(NULL,"shaders/texture");
14  t1=new texture(800,600,1);
15  t2=new texture(800,600,1);
16  t1->loadpng("test.png");
17  t1->ram2gpu();
18
19  for(int i=0;i<500;i++)
20  {
21    s->t[0]=t1;
```

```
22    s->tout[0]=t2;
23    s->bind();
24    s->step();
25
26    temp=t2;        // pointer swap for iteration
27    t2=t1;
28    t1=temp;
29   }
30
31   t2->gpu2ram();
32   t2->savepng("save.png");
33  }
```

Listing 17 Main source code example for a simple iteration process.

2.2 Example: shallow water equation

The last code example in this chapter applies the explained techniques from last two sections to give an implementation of a shallow water equation system on GPUs. For simplicity reasons the equations are implemented in the non conservative form, using a third order upwind for the spatial discretisation in the advection scheme and a three step Runge Kutta time integration method. The used advection and time integration schemes are also used in the model ASAMgpu and explained in more detail in the next chapter. Coriolis forces are neglected.

$$\frac{\partial h}{\partial t} = -\nabla \cdot (\mathbf{v}h) \tag{1}$$

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v}\nabla)\,\mathbf{v} - g\nabla h \tag{2}$$

The equation system describes the time evolution of a height field (h), wich is advected with the velocity field ($\mathbf{v}$) and the evolution of the velocity field that is advected as well and influenced trough the gravitational force if the height field is not in equilibrium with the adjacent cells. The system can be interpreted as a limited area of shallow water, where perturbations of the

height field will cause circular propagating waves in the velocity and height field (like raindrops in a puddle). Because of the length, the listings for the main source code and the shader kernel sources for the fragment shaders are given in the appendix (listings 20-24). The corresponding vertex shaders are not shown separately, because they are all the same and equal to the one from the texture example (listing 14). Figure 4 is an overview about the used textures and shaders and the flowchart of the main program including data flow through the shaders.

The code uses four texture pointers, one to save the old time step, one for the right hand side of the equations and one for the new time step. The fourth pointer is just a temporary help pointer during texture swap (line 98), so no additional memory is reserved for this one. The first channel of the data textures contains the current height of the surface and the second and third channel contain the velocities in x- and y-direction, the fourth channel is not used in this example. Respectively the first channel for the right hand side texture contains the source and sinks for the height and the second and third channel the sources and sinks for the velocities.

Beside the three textures and the swapping pointer the main source defines four shader kernels. The first one is the *s_plot* shader, used to draw the resulting data into the framebuffer to display it on the screen. To do that with a water like appearance the current height in a grid cell is mapped to a colorspace from blue to white. The next shader is the right hand side (RHS) shader (*s_rhs*) containing the calculations for the sources and sinks for all three used components, including the upwind scheme and gravity. The *s_step* shader object performs an Euler step, that means it multiplies the right hand side with a given time step and adds it to the incoming texture values. This shader object is used to implement the Runge Kutta scheme. The last shader is the *s_drop* shader, used to add a perturbation to the height field. This perturbation is added in the main function at a random position every 100th time step (line 50-58).

After the initialisation of the shaders and the texture fields, the mainloop starts. During this the pertubation is applied, which in this case is effectively just drawing one point with the defined pointsize directly into the height

field using the *s_drop* shader object. The next lines define the Runge Kutta scheme. Therefore at first the *s_rhs* is executed to calculate the forcings. Then a step forward in time with a third of the complete time step is done. With this intermediate state, new forcings are calculated and integrated in time from the original state up to half of the target time step. There again new forcings are evaluated and used for the complete time step giving an approximation for the new state. When all three Runge Kutta stages finished the new and old time step textures are swapped and the current state is plottet using the *s_plot* shader. One possible result after 500 frames is shown in figure 3.

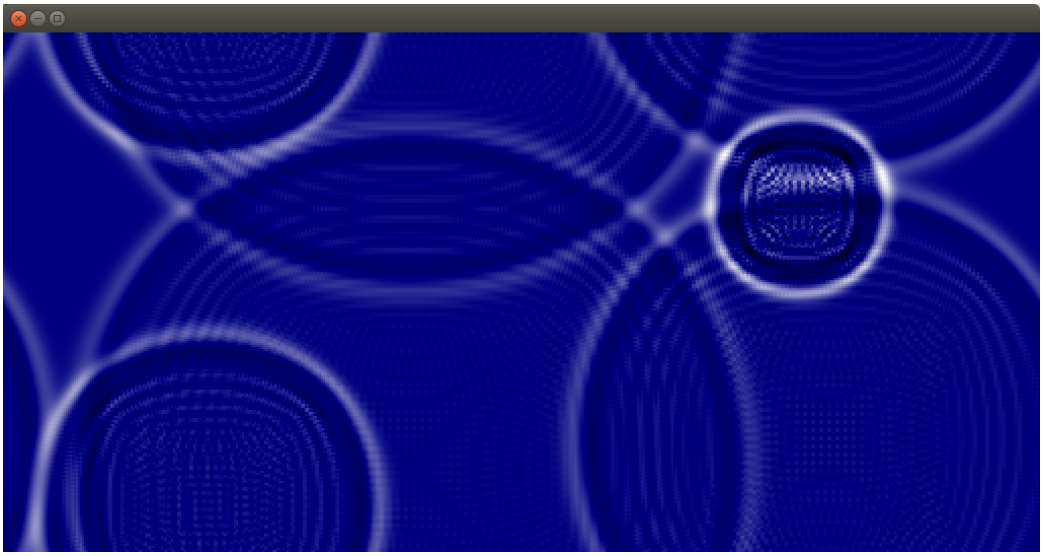

Figure 3 Simple visualisation of the shallow water example height field after 500 frames using the 2d color plot shader.

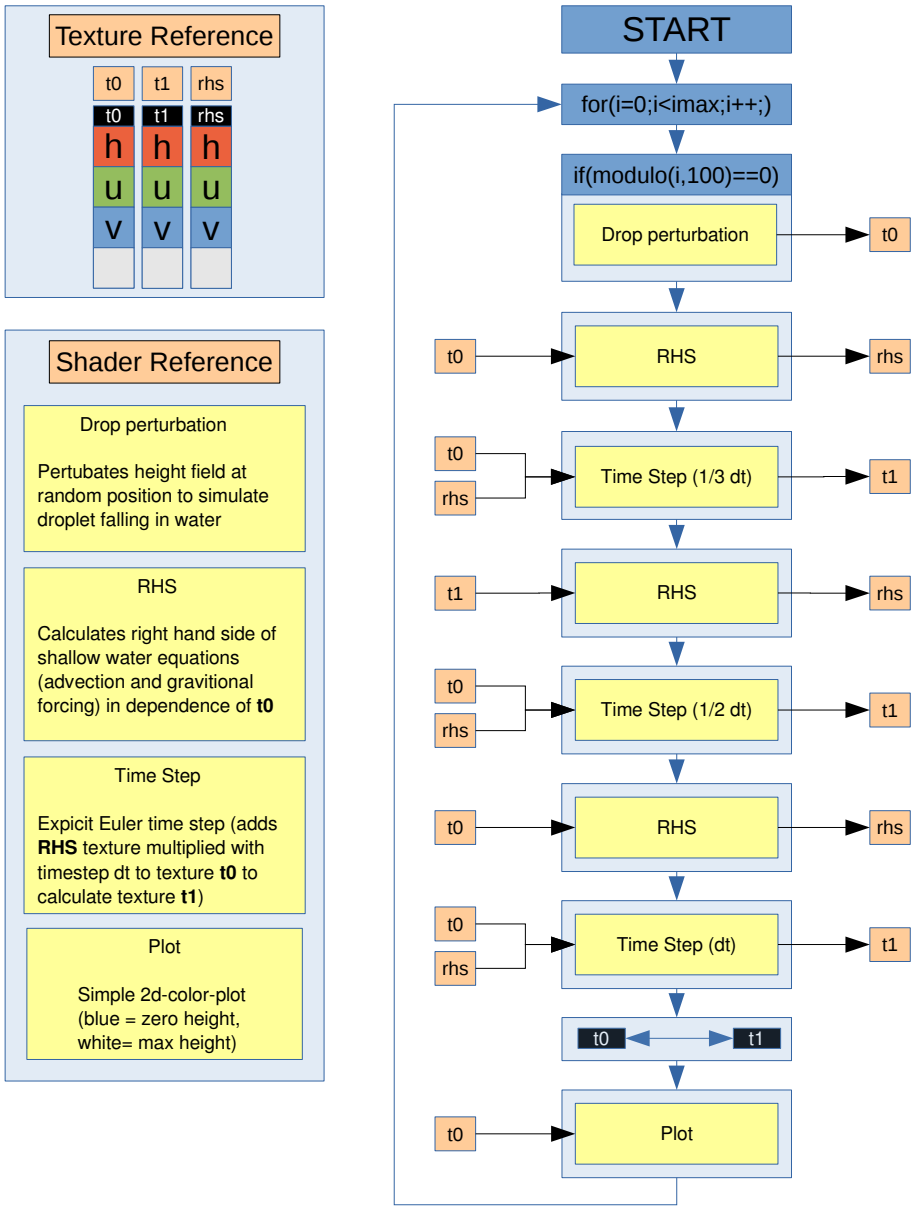

Figure 4 Flowchart, texture- and shader reference card for the simple shallow water example. The texture reference displays the used textures with the included components (height, velocity fields, fourth channel is unused). The shader reference gives a short idea of the shader task. The flowchart is a visualisation of the program and data flow with input textures on the left side and the output texture on the right side.

3 The Model ASAMgpu

This section describes the application of the developed shader and texture classes presented in the last chapter for a local to regional scale atmospheric model. Parts of this description including governing equations, the new entropy variable, the handling of the microphysical forcings and some of the examples in the next chapter were already published in Horn [2012].

The main interest during this development process was to gain understanding in high resolution cloud modeling including microphysical parameterizations that are suitable to study the influences of different aerosol concentrations on cloud properties and drizzle formation. At the same time the model had to be fast enough to perform calculations in three dimensional domains with an as large horizontal extend as possible to capture complete cloud systems. Because the focus lay on marine stratocumulus and shallow cumulus cloud decks without ice, the microphysical parameterisation is restricted to warm cloud processes.

The modelling process started with the implementation of a basic fully compressible computational flow model (CFD), including transport equations for density and momentum. In the next step a prognostic entropy variable and the according equation was added. The development then was continued by adding moisture variables, like water vapor and cloud water and the necessary equations to parameterize the phase transition from water vapor to cloud water. With that the term for the latent heat release had to be added in the entropy equation. Finally to study aerosol cloud interactions a two moment microphysical parameterisation for warm cloud processes was added including number densities for the different water phases as prognostic variables. The so developed model was then used to perform some theoretical experiments to verify the implemented processes and also one practical application is presented.

The model is written in C++, using OpenGL and GLSL. It is embedded in a web server environment using the Apache web server and PHP for remote development, runtime control and analysis. Two GPU server nodes were set up during the development. Both devices were assembled by Supermicro,

with space for four GPUs and Intel Xeon CPUs. The first node is equipped with four Xeon 5530 with four cores each, and the second one with four Xeon X5650 with six cores each. The model ASAMgpu does not benefit from the higher CPU core count, because it uses one thread per GPU, so never more then four threads were used. The first server is equipped with two ATI Radeon HD 4890 graphics cards with 1 GB memory per device and the second server node with four ATI Radeon HD 5870 with 2 GB memory each. Theoretical peak performance for the first node is 2.7 TFlops and the second reaches about 11 TFlops. In practice this values are decreased by communication costs through the relative slow PCI Express 2.0 bus. The final caluclations were performed with a single AMD R9 290 Tri-X with 4 GB memory and a theoretical peak performance of 5.12 TFlops.

3.1 Governing equations

The equations used in the GPU-Model are a form of the Euler equations for a compressible fluid in conservative form where the conservation of mass is applied for the bulk density (ρ). In addition further transport equations for the partial phases water vapor density (ρ_v), cloud water density (ρ_c) and rain water density (ρ_r), including the source terms from the microphysics ($S_{\rho_i MP}$) are used. The momentum equation is the standard Euler equation using bulk density and bulk momentum. The energy equation is written in the form of an entropy variable (σ) derived in section 3.4. In addition to the mass density transport equations, similar equations are included for available cloud condensation nuclei density (N_{CCN}), cloud droplet density (N_c) and rain droplet density (N_r), again with sources from the microphysical parameterization ($S_{N_i MP}$). These source terms include mass transfer between the different phases and changes in number densities due to the processes of condensation and evaporation, activation, selfcollection, autoconversion and sedimentation.

Subgrid scale turbulence, the Coriolis force and ice-phase microphysics

are currently ignored. So the basic equations can be written as follows:

$$\frac{\partial \rho}{\partial t} + \nabla (\mathbf{v}\rho) = 0 \tag{3}$$

$$\frac{\partial \rho_i}{\partial t} + \nabla (\mathbf{v}\rho_i) = S_{\rho_i MP} \qquad i = v, c, r \tag{4}$$

$$\frac{\partial N_i}{\partial t} + \nabla (\mathbf{v}N_i) = S_{N_i MP} \qquad i = CCN, c, r \tag{5}$$

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla (\rho \mathbf{v} \cdot \mathbf{v}) = -\nabla p - \rho \mathbf{g} \tag{6}$$

$$\frac{\partial \rho \sigma}{\partial t} + \nabla (\mathbf{v}\rho\sigma) = S_{\rho\sigma MP} \tag{7}$$

3.2 Spatial discretization

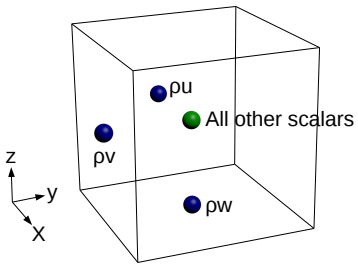

Figure 5 Illustration of the used Arakawa-C grid. The momentum variables are stored on the left face in the particular direction, all other scalars in the center of the gridcell.

The model uses a staggered grid (Arakawa-C, figure 5) with cell centered scalars and the velocity components stored at corresponding faces. For the advection scheme the scalars are interpolated to cell faces using a third order upwind scheme without limiters once every Runge Kutta intermediate step. The bottle neck for GPGPU is the PCIe bus, which is very slow compared to internal GPU communication, so that such a small stencil strongly increases performance on multi GPU setups. Listing 18 shows the equations for the fluxes at cell faces in the case of positive and negative wind velocity and the final flux divergence through advection in X-direction. The used variables

are illustrated in figure 6. The velocities *ulf* and *urf* are computed from the face centered momentum values divided by the arithmetic mean of the adjacent cell centered densities. *Fxlp* and *Fxrp* are the fluxes at the left and right cell face in case of positive wind speed at the according face, respectivly *Fxlm* and *Fxrm* are the fluxes in case of negative wind velocity. Hence all variables were initialized with zero, the result *Fx* is the final flux divergence in the cell for the advection in X-direction.

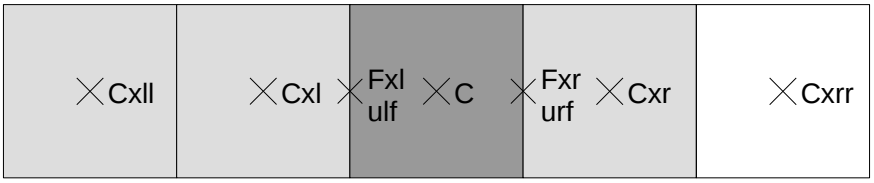

Figure 6 Illustration of the upwind stencil for $u > 0$, the variable c represents the currently advected scalar variable. F and u are the flux and the velocity at the corresponding face.

```
1  if(ulf>0.0)Fxlp=ulf*(  c/3.+5./6.*cxl-cxll/6.);
2  if(urf>0.0)Fxrp=urf*(cxr/3.+5./6.*  c- cxl/6.);
3  if(ulf<0.0)Fxlm=ulf*(cxl/3.+5./6.*  c- cxr/6.);
4  if(urf<0.0)Fxrm=urf*(  c/3.+5./6.*cxr-cxrr/6.);
5  Fx=(Fxlp-Fxrp+Fxlm-Fxrm)/dx;
```

Listing 18 Code snippet showing the third order upwind interpolation used in the advection scheme.

For the application of the upwind scheme during the evaluation of the momentum equation, the face centered values are shifted to cell centered values. This is done by a simple shader called *FaceToCell* which computes the arithmetic mean of the according momentums. The computed cell centered momentum values are stored and then processed similar to the advection step for the other scalar variables. The derived source values for momentum are now positioned in the cell center, so they have to be shifted back to face centered values. This happens again using the simple arithmetic mean of the sources in the adjacent cells.

3.3 Time integration

The time integration scheme follows Wicker and Skamarock [1998], using an explicit three step Runge Kutta scheme (RK3) with a time splitting algorithm for the fast pressure waves, which are integrated using a simple leapfrog algorithm. The Butcher tableau of the used low-storage RK3 is shown in figure 7. This RK3-Leapfrog combination has to be stabilized using divergence damping.

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{3} & \frac{1}{3} & & \\ \frac{1}{2} & 0 & \frac{1}{2} & \\ \hline & 0 & 0 & 1 \end{array}$$

Figure 7 Butcher tableau of the used Runge Kutta scheme

During every acoustic timestep pressure gradients and microphysical processes are computed. Similar time integration methods are commonly used by Bryan and Fritsch [2002], in the weather research forecast model (WRF, Skamarock and Klemp [2008]), and the COSMO by the german weather service (DWD).

3.4 A new thermodynamic variable

To describe atmospheric processes including heat fluxes, radiation and phase transitions, the set of equations presented in the last section include a prognostic equation describing the transport and the sources and sinks for energy. This can be done using different thermodynamic variables, e.g. the total energy, temperature, potential temperature, equivalent potential temperature or entropy. During this work a new variable was derived with some important and practical properties. At first it should be conserved during isentropic processes like advection, so no additional source terms containing gradients of vertical velocity or pressure appear in the transport equation in the case without a phase transition. Second, the absolute temperature and the pressure are needed for the computation of boundary conditions, microphysics and the evaluation of the pressure gradient in the momentum

equation, so those values should be explicitly computable. And the source terms in case of a phase transition should be derivable easily. This section shows the derivation of such a variable.

Starting with the first law of thermodynamics, written in specific quantities, including phase transition from water vapor to liquid water

$$du\left(s,\alpha,\rho_l,\rho_v\right) = Tds - pd\alpha + \frac{1}{\rho}\left(\mu_v - \mu_l\right)d\rho_v \tag{8}$$

with the definitions of enthalpy and latent heat

$$h = u + p\alpha \quad \text{and} \quad L_v = \left(\mu_v - \mu_l\right) \tag{9}$$

we get

$$dh = Tds + \alpha dp + \frac{1}{\rho}L_v d\rho_v \tag{10}$$

Introducing the definitions of specific heat capacity at constant pressure and the gas constant for a mixture,

$$dh = C_{pml}dT \tag{11}$$

$$C_{pml} = \frac{\rho_d c_{pd} + \rho_v c_{pv} + \rho_l c_{pl}}{\rho} \quad \text{and} \quad R_{ml} = \frac{\rho_d R_d + \rho_v R_v}{\rho} \tag{12}$$

and the equation of state for a vapor air mixture

$$\alpha = \frac{R_{ml}T}{p} \tag{13}$$

we get

$$C_{pml}dT = Tds + \frac{R_{ml}T}{p}dp + \frac{L_v}{\rho}d\rho_l \tag{14}$$

or

$$ds = \frac{C_{pml}}{T}dT - \frac{R_{ml}}{p}dp - \frac{L_v}{\rho T}d\rho_l. \tag{15}$$

Integration leads to

$$s = C_{pml} \ln(T) - R_{ml} \ln(p) - \int \frac{L_v}{\rho T} d\rho_l + C_0 \tag{16}$$

Introducing a quantity σ as a measure for entropy content caused by temperature and pressure

$$\sigma = C_{pml} \ln(T) - R_{ml} \ln(p) \tag{17}$$

leads to

$$s = \sigma - \int \frac{L_v}{\rho T} d\rho_l + s_0 \tag{18}$$

For the phase transition from vapor to liquid water the assumption of conservation of total mass ($\frac{d\rho_d}{dt} = 0; \frac{d\rho_v}{dt} = -\frac{d\rho_l}{dt}$) yields

$$\begin{aligned} \frac{ds}{dt} &= \frac{d\sigma}{dt} - \frac{1}{\rho}\left(C_{pl} - C_{pv}\right) \ln(T) \frac{d\rho_l}{dt} - \frac{1}{\rho} R_v \ln(p) \frac{d\rho_l}{dt} \\ &\quad - \frac{L_v}{\rho T} \frac{d\rho_l}{dt} \end{aligned} \tag{19}$$

with

$$\frac{ds}{dt} = \frac{1}{T} \delta Q \tag{20}$$

we get the evolution equation for $\rho\sigma$

$$\begin{aligned} \left(\frac{\partial \rho\sigma}{\partial t} + \nabla\left(\mathbf{v}\rho\sigma\right) \right) &= \frac{\rho}{T} \delta Q + \left(C_{pl} - C_{pv}\right) \ln(T) \frac{d\rho_l}{dt} \\ &\quad + R_v \ln(p) \frac{d\rho_l}{dt} + \frac{L_v}{T} \frac{d\rho_l}{dt} \end{aligned} \tag{21}$$

The so derived quantity has the desired advantages. First it is conserved under adiabatic processes without phase transitions. Second, explicit equations for absolute pressure and temperature can be obtained.

$$T = \exp\left(\frac{\sigma}{C_{pml} - R_{ml}} + \frac{\ln(\rho R_{ml})}{C_{pml}/R_{ml} - 1}\right) \quad (22)$$

$$p = \exp\left(\frac{\sigma}{C_{pml} - R_{ml}} + \frac{\ln(\rho R_{ml})}{1 - R_{ml}/C_{pml}}\right) \quad (23)$$

And last but not least the source terms in the case of a phase transition in equation 21 are very simple.

3.5 Microphysics

The model includes a two moment microphysics, based on the work of [Seifert and Beheng 2006] (SB2005), but not all processes described by SB2005 are included. Processes currently implemented are activation of cloud condensation nuclei to cloud droplets, condensation and evaporation of water vapor to/from these droplets, selfcollection of cloud droplets, autoconversion from cloud droplets to rain droplets, selfcollection of rain droplets, accretion of cloud droplets by rain, sedimentation and evaporation of rain drops. Collisional breakup and ice-phase microphysics as very important processes in deep convective clouds are neglected, because the focus was on shallow cumulus convection. The advection without limiters and numerical errors during the condensation/evaporation process causes unphysical small negative values in the prognostic mass and number density variables, so all negative densities have to be clamped to zero for the parameterizations after SB2005. All transition rates are processed by an additional limiter which ensures that negative values will be drawn back to zero and transition rates will not exceed available quantities. For this limiter the unclamped values have to be used.

3.5.1 Limiter example: activation

Detailed description for the microphysical source terms can be found in SB2005. In the ASAMgpu model these source terms are modified to increase robustness and to handle negative values caused by the third order

advection scheme without flux limiter. In this section this algorithm is presented using the example of activation from cloud condensation nucleis to cloud droplets. After SB2005 the activation rates are nonzero if the cell is supersaturated ($S > 0.0$), the vertical velocity is positive ($w > 0.0\,\mathrm{m/s}$), the gradient of supersaturation in vertical direction is positive ($dS/dz > 0.0\,\mathrm{m}^{-1}$) and the temperature is above $233.15\,\mathrm{K}$ ($T > 233.15\,\mathrm{K}$). In this context S is the supersaturation in percent. In SB2005 a power law is used as an empirical activation spectra to compute available CCN number density as a function of supersaturation and a background density. In the ASAMgpu model this background density N_{ccn} of available cloud condensation nuclei is an advected prognostic variable as well. With that the activation rate is given by

$$\frac{\partial N_{cSB}}{\partial t} = N_{ccn} k_{ccn} \frac{1}{S} \frac{dS}{dz} w \tag{24}$$

In the ASAMgpu model this activation rate gets limited by the unclamped available cloud condensation nuclei density (N_{ccn}) using a form of the triangle inequality (Eq. 26). The result of this limiting procedure is near the (*for*)cing if the (*lim*)iter is much larger then the forcing. This is the case if a huge quantity of cloud condensation nuclei is available. If the forcing is near the limiter, the process is damped to not consume more then the available N_{ccn}. But if the limiter is negative the equation always gives a result that compensates the negative values and draws them back to zero, while conserving mass and number density budgets.

$$for = \frac{\partial N_{cSB}}{\partial t} \tag{25}$$

$$lim = N_{ccn} \tag{26}$$

$$\frac{\partial N_{ccn}}{\partial t} = for + lim - \sqrt{for^2 + lim^2} \tag{27}$$

The mass change from water vapor to cloud water can be calculated assuming all activated droplets contain an arbitrary small drop mass of $10^{-12}\,\mathrm{kg}$. The condensed water mass determines the source from latent heat and the change in $\rho\sigma$ caused by the change of mass fractions in the gas constant and heat

capacity of the mixture.

$$\frac{\partial \rho_c}{\partial t} = 10^{-12}\text{kg}\frac{\partial N_c}{\partial t} \tag{28}$$

$$\frac{\partial \rho\sigma}{\partial t} = L_v \frac{1}{T}\frac{\partial \rho_c}{\partial t} + \ln(T)\left(C_{pl} - C_{pv}\right)\frac{\partial \rho_c}{\partial t} \tag{29}$$

$$+ \ln(p)\, R_v \frac{\partial \rho_c}{\partial t} \tag{30}$$

3.5.2 Condensation and evaporation of cloud water

The saturation adjustment technique is a common method, to calculate the amount of water vapor condensed during a timestep. Therefore all microphysical parameterizations are processed and after that the complete fraction of water vapor above the saturation level is handled as condensate. In this approach supersaturation is reduced immediately and the time integration for the microphysics has to be processed separately. In the ASAMgpu model the saturation adjustment technique was replaced by a relaxation process from vapor pressure to saturation vapor pressure. The forcing for this process is the difference between actual water vapor density and the water vapor density at saturation point. The process is limited by the available cloud water, so condensation occurs at supersaturation and evaporation occurs if the gridcell is unsaturated and cloud water is available. The time scale of this process is controlled by a constant C_{cond}. If this constant is very high, the scheme is near the saturation adjustment technique and supersaturation will be decreased nearly instantly. With a very low constant this process gets too slow and convection may be suppressed, in this case the moist bubble example presented later will produce wrong results. A constant set to 0.1 seems to be a good choice.

$$for = \rho_v - (p_{vs}T/R_v) \tag{31}$$

$$lim = \rho_c \tag{32}$$

$$\frac{\partial \rho_c}{\partial t} = C_{cond}\left(for - lim + \sqrt{for^2 + lim^2}\right) \tag{33}$$

Again from the condensed/evaporated mass the necessary source/sink from latent heat release/consumption for the entropy variable has to be derived.

$$\frac{d\rho\sigma}{dt} = \left(L_v \frac{1}{T} + \ln(T)\left(C_{pl} - C_{pv}\right) + \ln(p)\, R_v \right) \frac{d\rho_c}{dt} \tag{34}$$

Two more simple equations are used to ensure that, if no condensate exists, droplet number density reduces to zero, or if condensate exists droplet number density is within the limits defined by the distribution parameters (see SB2005). The speed of this correction is controlled by the constant C currently set to $0.01 s^{-1}$. This process is a transition between available cloud condensation nuclei number density and cloud droplet number density, so evaporated droplets produce new possible cloud condensation nuclei. These corrections are reducing N_c if droplets get to small according to appendix D in SB2005.

$$\frac{dN_c}{dt} = \min\left(\quad 0, \quad C\left(\frac{\rho_c}{x_{min}} - N_c \right) \right) \tag{35}$$

and increasing N_c if droplets get to big

$$\frac{dN_c}{dt} = \max\left(\quad 0, \quad C\left(\frac{\rho_c}{x_{max}} - N_c \right) \right) \tag{36}$$

with

$$\frac{dN_{ccn}}{dt} = -\frac{dN_c}{dt} \tag{37}$$

4 Applications

This section presents applications simulated with the ASAMgpu model. The first four examples are simple setups with the focus on testing the model dynamics and the basic functionality of the microphysical implementation. The first example is a dry convection between two plates, with a positive sensible heat flux at the bottom surface and a negative one at the top. The next test case is a rising heat bubble in a dry atmosphere. The third case is a cold pool falling to the ground, inducing a strong density current, and the fourth simple case is a heat bubble in a moist environment slightly above the saturation point, where the latent heat released during condensation is a crucial factor for the bubble to rise to the final height. To test the complete microphysics including selfcollection, autoconversion, sedimentation and evaporation of rain marine stratocumulus and shallow cumulus cloud layers are good examples. The Global Atmospheric System Studies (GASS, formerly known as GCSS) boundary layer cloud group which is part of the Global Energy and Water Exchanges Project (GEWEX Cloud System Science Team [1993]) formulated some interesting test cases, used for LES model inter comparisons. Three of those, the BOMEX (Siebesma et al. [2003]), DYCOMS (Stevens et al. [2005]) and RICO (VanZanten et al. [2011]) case, were choosen to evaluate the microphysical performance of the model ASAMgpu. The simulations with a typical domain size of $256 \times 256 \times 64$ cells at a resolution of 60 m, a time step of 1 s with 18 acoustic steps and a simulation time of 24 hours could be realized within 16 hours computation time on one single GPU. As a more practical oriented application the ASAMgpu was also used to investigate the influence of an island on the structure of the marine boundary layer during the SAMUM-II campaign on Cap Verde Island (Engelmann et al. [2011]).

4.1 Dry thermally driven boundary layer

As an example for a dry boundary layer test case, the setup consists of two imaginary very large plates, where the bottom one was heated and the top plate was cooled down. Both with equal constant sensible heat fluxes like shown in figure 8.

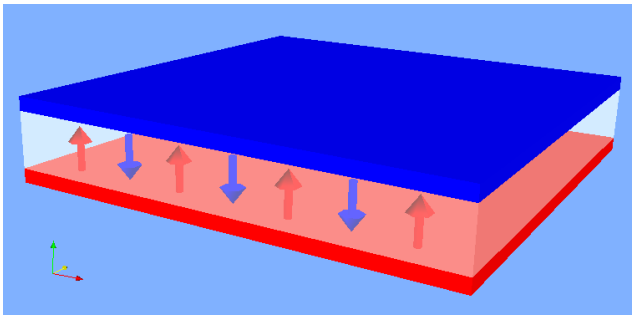

Figure 8 Schematic view of the test environment for the thermally driven boundary layer.

If an initial break of symmetry is model inherent a convective boundary layer evolves, otherwise the initial symmetry has to be broken by an artificial perturbation. This may be a noise or a still symmetric perturbation. The three examples show three dimensional domains, with a surface heat flux at the bottom of $50\,\mathrm{Wm}^{-2}$ and respectively $-50\,\mathrm{Wm}^{-2}$ at the top of the domain. The initial perturbation in the first case is a small temperature derivation of $10^{-3}\,\mathrm{K}$ in the center cell of the domain. In the second case, a regular pattern of sixteen equally distributed cells were perturbed. Finally for the third case a random noise was applied to the density field in the boundary layer. The domain size for the three dimensional simulation was $256 \times 256 \times 16$ cells at a spatial resolution of $250 \times 250 \times 50\,\mathrm{m}$ and an advective time step of one second.

Figure 9 shows the vertical velocity for a horizontal cutplane at the height of 400 m above surface for the case with one single initial perturbation in the center of the domain. At the beginning of the simulation the first layer at the bottom gets heated and the upper most layer gets cooled. This induces pressure waves. In the case without perturbation, these pressure waves are planar waves traveling up and down in the domain, resulting in an expansion of the air at the lower surface and a contraction of the cooled air at the top. Hence the heating and cooling is isotropic, the horizontal pressure gradient is zero everywhere and no horizontal motions evolve. The change in density through cooling and heating produces vertical motions only.

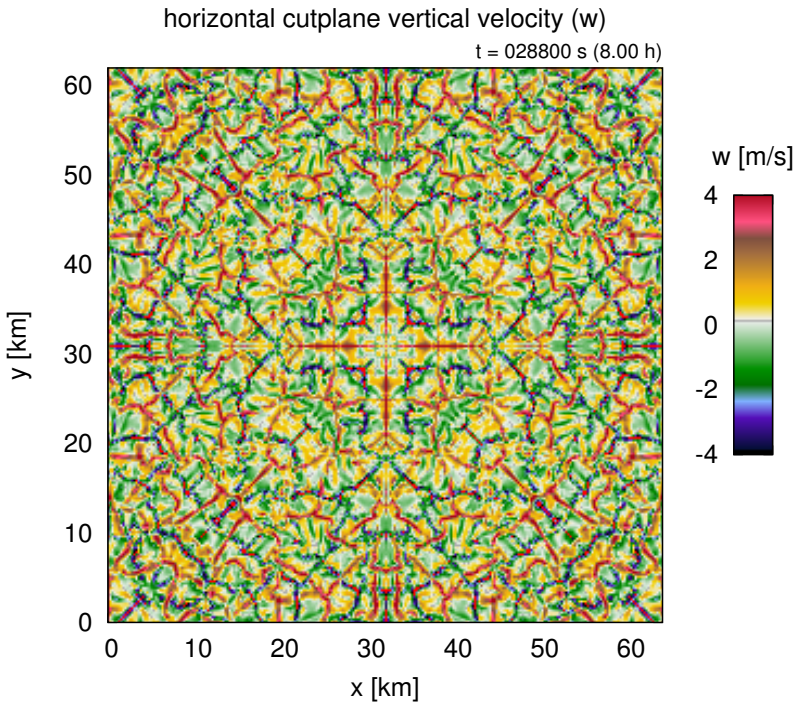

Figure 9 Horizontal cutplane at half of the domain height for the vertical velocity for one central initial perturbation after 8 hours.

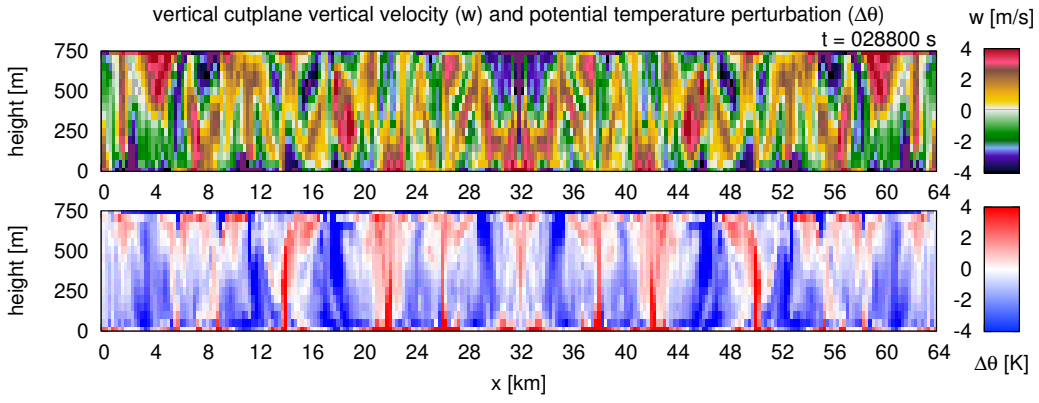

Figure 10 Vertical cutplane for the vertical velocity and potential temperature deviation for one central initial perturbation after 8 hours.

In the presence of a perturbation, small pressure gradients cause spherically propagating fluctuations in the horizontal and vertical velocity fields. At the heated bottom layer even a small convergence of warm air is a self amplifying process. If a small updraft of warm air exists, it reduces pressure at the lowest layers resulting in a convergence of warm air to resupply the updraft. It is sustained or intensified as long as warm air is available. A symmetric process causes cold downdrafts in the cooled layer at the top of the domain for the air with lower temperature and higher density. The amount of energy and mass transported from the bottom to the higher layers has to be in balance with the amount transported down from the top. The overall mean temperature of the system remains constant, nevertheless a fraction of the energy fluxe is converted to turbulent motion accelerating the circulation between the two plates. The grid size, the geometry of the domain in combination with the initial perturbation and the numerical diffusion determine the resulting pattern. With time the cells grow and high frequency oscillations are damped by the numerical diffusion of the upwind advection scheme. Larger cells grow more rapidly and acquire the supply of warm air, which then is missing for smaller cells resulting in their dissolution.

Figure 11 shows the same cutplane as in the single perturbation case but with sixteen equally distributed small initial perturbations. The initial symmetry is conserved and in contrast to the large convection cell in figure 9 sixteen smaller equal convection cells occur. The induced symmetry patterns are conserved for the complete model runtime.

For the last simulation again the vertical velocity in the two cutplanes are shown in figure 13 and 14. But in contrast to the last both examples this time the initial perturbation is a small random temperature derivation below 10^{-3} K. The results for the simulation with the applied random noise appear quite similar to structures that can be observed in simulations of inversion topped boundary layers as well. The conservation of symmetry in the first two examples is an indicator for the correct numerical implementation of the advection and the pressure gradients in the momentum equation.

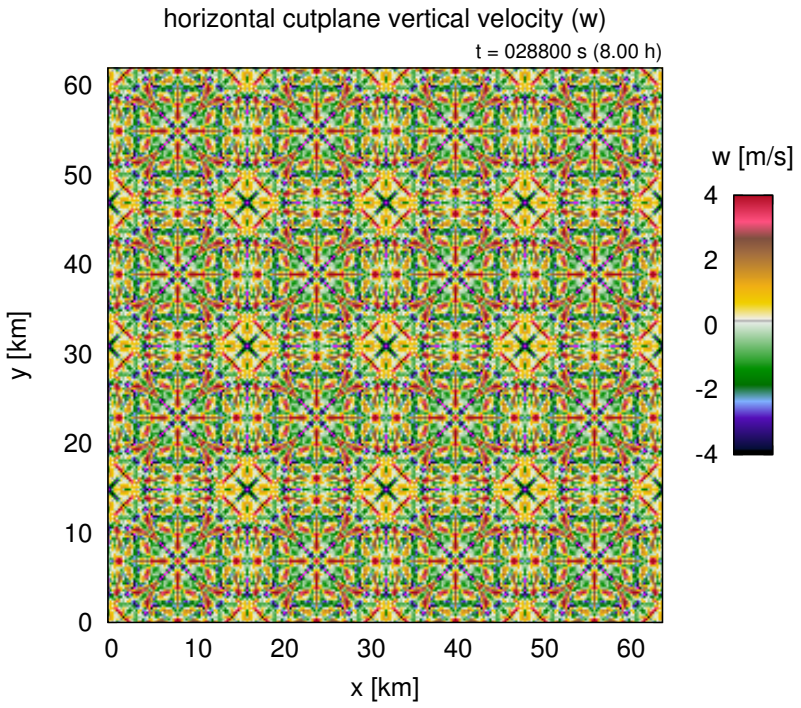

Figure 11 Horizontal cutplane at half of the domain height for the vertical velocity for sixteen symmetric initial perturbations after 8 hours.

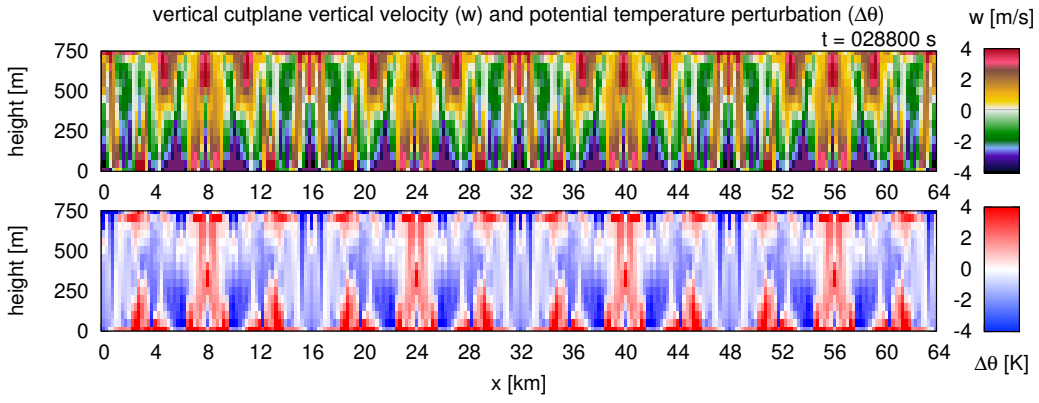

Figure 12 Vertical cutplane for the vertical velocity and potential temperature deviation for sixteen symmetric initial perturbations after 8 hours.

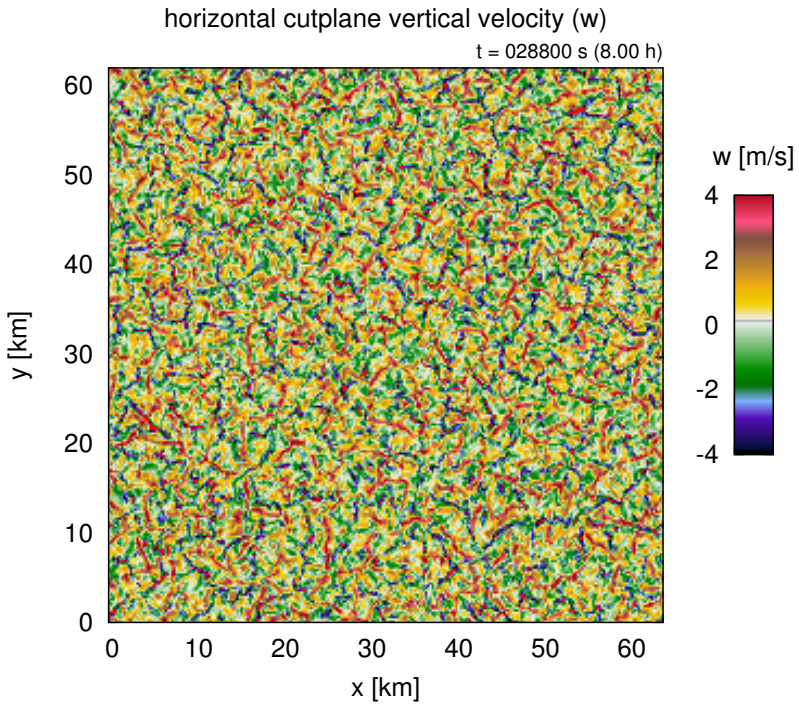

Figure 13 Horizontal cutplane for the vertical velocity at half of the domain height after 8 hours. In this case the initial state was perturbed with a small random noise in the temperature field.

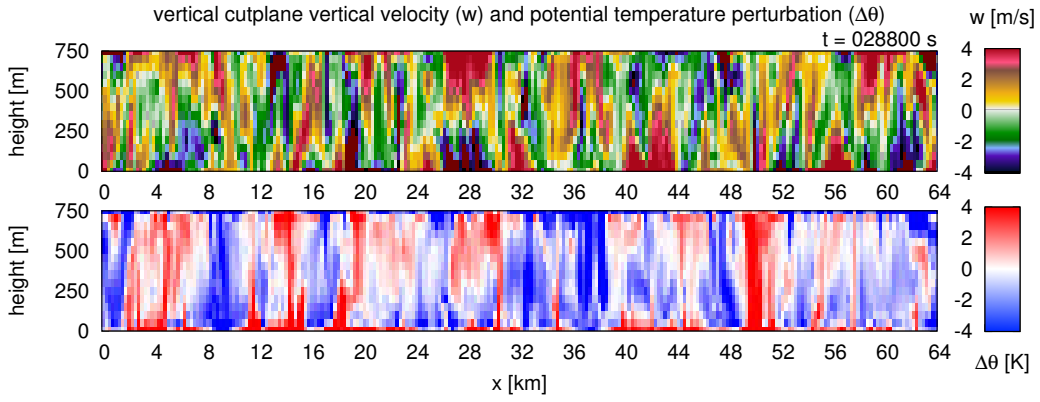

Figure 14 Vertical cutplane for the vertical velocity and potential temperature deviation after 8 hours for the randomly perturbed case.

4.2 Dry heat bubble

The second test is a rising heat bubble under dry conditions embedded in a uniform horizontal flow field (Bryan and Fritsch [2002]). This test focuses on the representation of the advection terms and the correct evaluation of the buoyant forces in the momentum equation. The domain size is 160×80 cells at a spatial resolution of 125 m with periodic horizontal boundary conditions. The initial state consist of an adiabatic atmosphere with a perturbation in the potential temperature. The amplitude of the perturbation is 2 K with a radius of 2 km. The bubble is located in the horizontal center of the domain at 2 km height described by

$$x_c = 10000\,\mathrm{m} \quad z_c = r_x = r_z = 2000\,\mathrm{m} \tag{38}$$

$$L = \sqrt{\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{z - z_c}{r_z}\right)^2} \tag{39}$$

$$\theta'[\mathrm{K}] = 2\cos^2\left(\frac{\pi L}{2}\right) \tag{40}$$

A uniform horizontal velocity of 20 m/s is applied, leading to a transport of the bubble through the whole domain and the boundaries. After 1000 s a complete cycle is fulfilled and the bubble reaches the center of the domain again. The time steps for this test case were chosen after Jebens et al. [2009] with 2 s and 10 fast pressure steps and 7 s and 30 pressure steps as well. Divergence damping with a damping coefficient of $\nu = 0.025$ is used. The results for both time steps are equal and shown in figure 15. The bubble rises up until the top of the bubble reaches a height of 8 km. As expected the solution keeps more or less symmetric, small deviations are caused by the horizontal background flow which causes different horizontal velocity amplitudes in the right and the left circulating part of the bubble and with that also different numerical diffusion.

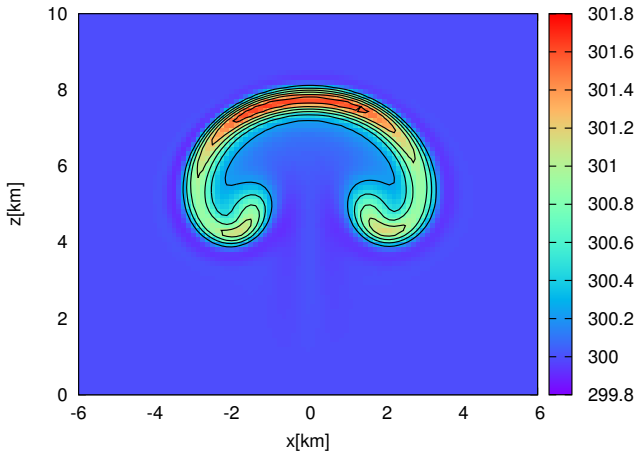

Figure 15 Result for the heat bubble test case after 1000 s (contours: pot. temp. 0.25 K).

4.3 Dry cold bubble

The third test case by Straka et al. [1993] has a slightly larger domain than the second one and in contrast to the heat bubble the perturbation now consist of a cold pool. During the simulation the cold pool descends until it reaches the surface where the cold air starts to spread in the horizontal direction. Two symmetric outflow boundary jets develop with high wind speeds moving in opposite direction with several typical vortex structures evolving. Those high wind speeds are a good test for the stability properties of the used time integration scheme. Again the initial state is a dry, adiabatic atmosphere with a uniform horizontal velocity field of 20 m/s. The domain now has 180×80 cells at a resolution of 200 m. With that one cylce needs 900 s. The bubble is initialized in the horizontal center of the domain again, but now at 4 km height, and the horizontal radius is extended to 4 km as well. The amplitude of the pertubation is 15 K and applied to the temperature described by:

$$x_c = 18000\,\mathrm{m} \quad z_c = r_x = 4000\,\mathrm{m} \quad r_z = 2000\,\mathrm{m} \tag{41}$$

$$L = \sqrt{\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{z - z_c}{r_z}\right)^2} \tag{42}$$

$$T'[\mathrm{K}] = -15\cos^2\left(\frac{\pi L}{2}\right) \tag{43}$$

The large timestep is 2 s and with that six pressure time steps are needed at this spatial scale to satisfy the Courant Fredrich Levy criteria for the acoustic waves. Again, divergence damping is necessary with a damping coefficient of $\nu = 0.025$. The result after one complete cycle is shown in figure 16. Compared to the results of Jebens et al. [2009] and Wicker and Skamarock [2002] the overall structure is reproduced, but the solution is more diffusive. This is caused by the implemented 3rd order upwind advection scheme, in contrast to the 5th order advection schemes used by the cited studies.

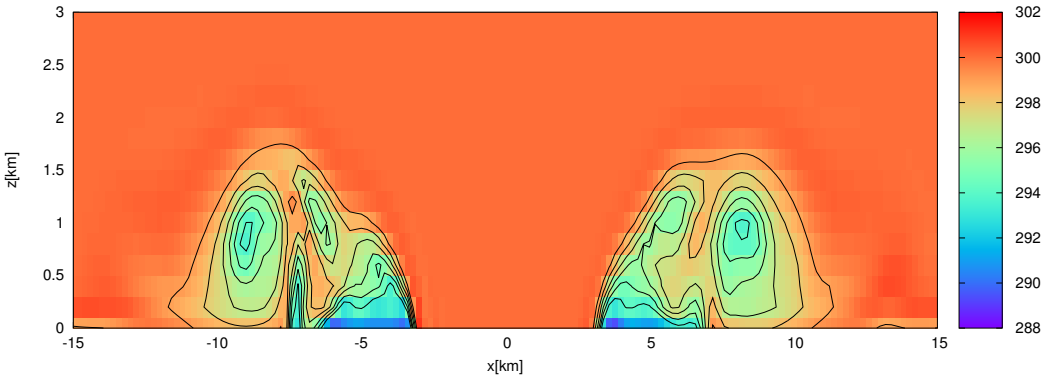

Figure 16 Result for the cold bubble test case after 900 s (contours: pot. temp. 1K).

4.4 Moist heat bubble

The fourth test case for the GPU model is a modification of the rising heat bubble. It was suggested by Bryan and Fritsch [2002], and is also a test for a part of the microphysics. In this case the initial state is again a hydrostatic

atmosphere with neutral stability for moist air. To simplify the definition of neutral stability for the moist case two assumptions are made. The first one is that the total water mixing ratio is constant and the second one is that all phase changes are exactly reversible. Under these assumptions a moist neutral atmosphere can be defined by a constant wet equivalent potential temperature, therefore the atmosphere has to be saturated at all levels. For this test the microphysical parameterisation is reduced to the reversible phase change, respectively only the processes of activation and condensation/evaporation are enabled. The simulation is a good proof for the condensation rates and the amount of latent heat release. If those are to slow the bubble stops rising and will not reach the final height of 8 km after 1000 s. All other parameters are similar to the dry heat bubble test case, that includes a slightly different resolution than in the work of Bryan and Fritsch (125 m vs 100 m), an advective timestep of 7 s with six steps for the acoustic modes and 20 m/s horizontal wind. The perturbation are added to a 320 K background equivalent potential temperature and are described by the following formulars.

$$x_c = 10000\,\mathrm{m} \quad z_c = 2000\,\mathrm{m} \quad r_x = r_z = 2000\,\mathrm{m} \tag{44}$$

$$L = \sqrt{\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{z - z_c}{r_z}\right)^2} \tag{45}$$

$$\theta'[\mathrm{K}] = 2\cos^2\left(\frac{\pi L}{2}\right) \tag{46}$$

The simulation results, shown in figure 17, should reproduce the structure and the lifting height of the dry heat bubble, but now in the potential liquid water temperature. Except for slight over- and undershoots caused by the advection scheme the results are in good agreement with the work of Bryan and Fritsch [2002].

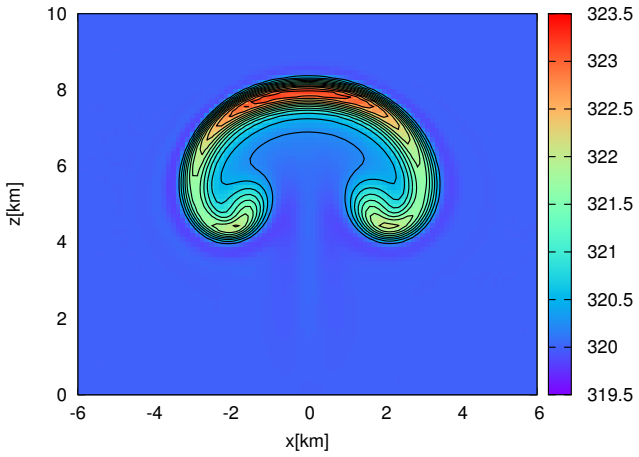

Figure 17 Result for the moist heat bubble test case after 1000 s (contours: equiv. pot. temp. 0.5K).

4.5 Trade Cumulus: BOMEX

The first more complex test case simulated with the ASAMgpu model framework is based on the Barbados Oceanographic a Meteorological Experiment (BOMEX) from the GASS LES inter comparison cases for shallow cumulus cloud convection. This experiment was carried out in 1969 with the objective to determine surface exchange fluxes between the ocean surface and the lower atmosphere. Later, based on the measurements, different LES studies, for example from Jiang and Cotton [2000], Heus et al. [2010] and a LES inter comparison from Siebesma et al. [2003], were realized. The surface fluxes are prescribed as fixed with $8\,\mathrm{W/m^2}$ sensible and $150\,\mathrm{W/m^2}$ latent heat flux. The wind speed is constant at $-8.75\,\mathrm{m/s}$ in the mixing layer up to a height of 700 m and then increasing at a rate of $1.8\,\mathrm{m/s}$ per kilometer.

$$\begin{aligned} u[\mathrm{m/s}] &= -8.75 && for \quad 0 < z < 700\,\mathrm{m} \\ u[\mathrm{m/s}] &= -8.75 + 1.8 \cdot 10^{-3}(z - 700) && for \quad z > 700\,\mathrm{m} \end{aligned} \tag{47}$$

The background profiles for the total water content and the liquid potential temperature are given as follows:

$$
\begin{array}{rcll}
 & & & q_t[\mathrm{g/kg}] = \\
0 < & z & <520 & 17.0 + (16.3 - 17.0)/(520) \cdot z \\
520 < & z & <1480 & 16.3 + (10.7 - 16.3)/(1480 - 520) \cdot (z - 520) \\
1480 < & z & <2000 & 10.7 + (\ 4.2 - 10.7)/(2000 - 1480) \cdot (z - 1480) \\
 & z & >2000 & 4.2 - 1.2 \cdot 10^{-3} \cdot (z - 2000)
\end{array}
\tag{48}
$$

$$
\begin{array}{rcll}
 & & & \theta_l[\mathrm{K}] = \\
0 < & z & <520 & 298.7 \\
520 < & z & <1480 & 298.7 + (302.4 - 298.7)/(1480 - 520) \cdot (z - 520) \\
1480 < & z & <2000 & 302.4 + (308.2 - 302.4)/(2000 - 1480) \cdot (z - 1480) \\
 & z & >2000 & 308.2 + 3.65 \cdot 10^{-3} \cdot (z - 2000)
\end{array}
\tag{49}
$$

Further applied large scale forcings are a subsidence, a radiative cooling of $2\,\mathrm{K\,day^{-1}}$, and a large scale advection term which emulates the transport of dry air into lower boundary layers

$$
\begin{array}{rcll}
 & & & w_{subsidence}[\mathrm{m/s}] = \\
0 < & z & <1500 & -(0.0065/1500) \cdot z \\
1500 < & z & <2100 & -0.0065 + 0.0065/(2100 - 1500) \cdot (z - 1500) \\
 & z & <2100 & 0.0
\end{array}
\tag{50}
$$

$$
\begin{array}{rcll}
 & & & d\theta/dt[\mathrm{K/s}] = \\
0 < & z & <1500 & -2.315 \cdot 10^{-5} \\
1500 < & z & <2500 & -2.315 \cdot 10^{-5} + 2.315 \cdot 10^{-5}/(2500 - 1500) \cdot (z - 1500) \\
 & z & >2500 & 0.0
\end{array}
\tag{51}
$$

$$
\begin{array}{rcll}
 & & & dq_t/dt[\mathrm{g/kg\,day^{-1}}] = \\
0 < & z & <300 & -1.2 \cdot 10^{-8} \\
300 < & z & <500 & -(1.2 \cdot 10^{-8} - 1.2 \cdot 10^{-8} \cdot (z-300)/(500-300)) \\
 & z & >500 & 0
\end{array}
\tag{52}
$$

The BOMEX test case reaches a steady dynamic state with clouds evolving and evaporating without recognizeable influence of precipitation. The cloud base is at 600 m and cloud top is between 1700 m and 2000 m. To break the symmetry of the setup, small perturbations are added to the potential temperature and the total water content. Those perturbations have an amplitude of 0.1 K for the temperature and $2.5 \cdot 10^{-2}$g/kg for the moisture.

The domain size for this simulation consists of $256 \times 256 \times 64$ cells at an isotropic resolution of 50 m. An advective time step of one second with 18 acoustic steps was chosen.

Figure 18 shows domain averaged values for the liquid water path (LWP), the cloud fraction and the vertically integrated turbulent kinetic energy (TKE). The begin of the simulation is dominated by the spin up process, where the yet non existing vertical velocities have to evolve versus the shear in the horizontal wind speed. While this happens moisture and heat accumulates in the lower layers of the boundary layer. When the buoyant forces get strong enough convection organizes and the accumulated heat and moisture reaches the condensation level. A strong peak in liquid water path is observable during that spin up process. In comparison to the other BOMEX simulations, that took part in the inter comparison project, this peak is stronger in the ASAMgpu, and it needs about 3 to 4 hours to reach the steady state. This can be explained through the non existing explicit subgrid turbulence model. Such a turbulence model may accelerate the erosion of wind shear and enhance the diffusion. This leads to less concentrated energy in the surface layer and hence a reduced spin up time.

After the spin up phase a more stable convection develops. This consists of rising heat and moisture bubbles, reaching condensation level at 600 m.

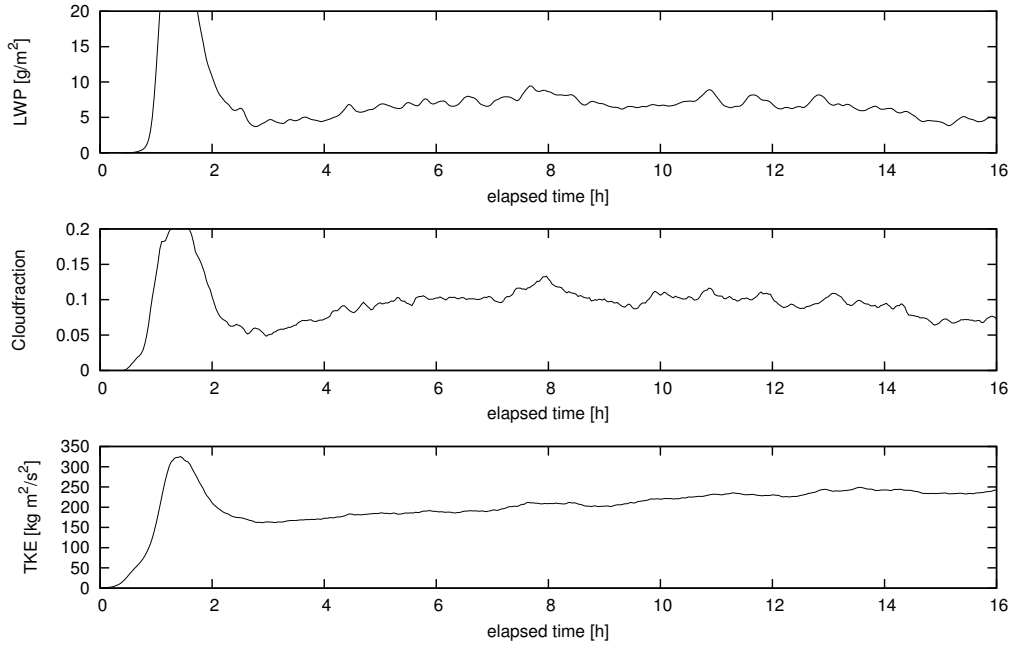

Figure 18 Domain averaged liquid water path (LWP), cloud fraction and vertically integrated turbulent kinetic energy (TKE) of the first 16 hours of the BOMEX test case.

At this height supersaturation occurs and activation of cloud droplet nucleis and condensation sets in. Latent heat release allows a further rising of the now existing cloud up to the inversion between 1500 m and 2000 m. During that the cloud gets slightly sheared with the mean wind and dryer air is entrained at the downstream side resulting in evaporation and cooling. A few well organized updrafts reach maximum heights of 1900 m.

The liquid water path and the turbulent kinetic energy show a very slight increase over the complete 16 hours simulation time while the cloud fraction remains constant. The liquid water path reaches values between $10\,\mathrm{g/m^2}$ and $15\,\mathrm{g/m^2}$. The cloud cover ranges from 0.1 up to 0.2. All values are in good agreement with the results of Siebesma et al. [2003] (figure 19) although the liquid water path is a bit overestimated. The resolved turbulent kinetic energy is with $250\,\mathrm{kgm^2/s^2}$ slightly lower compared to about $400\,\mathrm{kgm^2/s^2}$ at Siebesma et al. [2003]. This is also recognizeable in the comparison of the vertical profiles of the turbulent kinetic energy in figures 20 and 21.

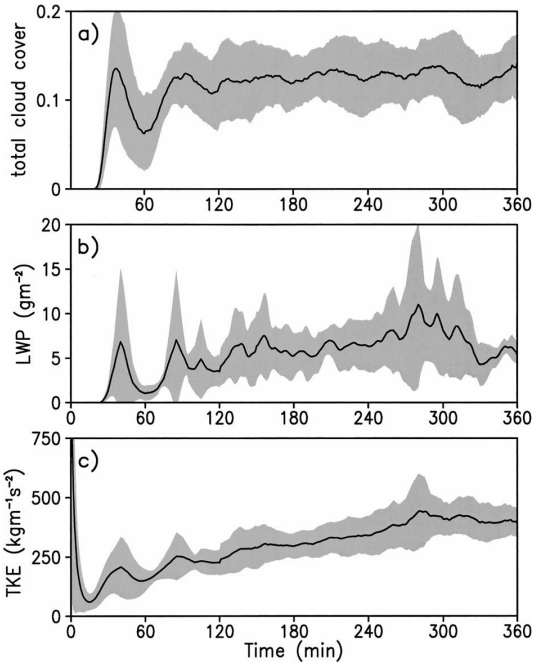

Figure 19 Time evolution of the first 6 hours of the BOMEX test case after Siebesma et al. [2003].

One possible reason for this is the quite diffusive third order upwind scheme for advection which dissipates higher frequency oscillations and eddies. The strongest deviation is located near the surface, where the TKE from Siebesma et al. [2003] shows values up to $0.4\,\mathrm{m}^2/\mathrm{s}^2$ while the maximum TKE value in the model ASAMgpu is at $0.25\,\mathrm{m}^2/\mathrm{s}^2$ (see figure 20 and 21). Above the near surface maximum both models show a decrease of TKE above 500 m and a second peak at 1500 m. Again the second peak in ASAMgpu is not as strong as the median from Siebesma et al. [2003]. Also the distribution of the liquid water content with height differs from the ensemble mean. The model ASAMgpu produces a stronger peak with 0.010 g/kg at a height of about 900 m and then decreases to $0.003\,\mathrm{g}/\mathrm{m}^3$ at 1600 m. This peak can not be found in the profile from Siebesma et al. [2003], where a more homogeneous distribution of the liquid water content with height is presented. The work of Slawinska et al. [2011] showed similar vertical profiles for cloud fraction and cloud droplet density like the ASAMgpu model, if in cloud activation

of cloud condensation nuclei is suppressed. The referred work uses a two moment warm microphysical scheme by Morrison and Grabowski [2008]. The profiles in Slawinska et al. [2011] change to the more isotropic behaviour with enabled in cloud droplet activation. The background density of potential cloud condensation nuclei for the ASAMgpu runs is initialized with $100\,\mathrm{mg}^{-1}$. This leads to a maximum of domain averaged cloud droplet density of $1.5 \cdot 10^6/\mathrm{m}^3$. Together with a cloud fraction of 0.08, which is the number of cells with more than $0.01\,\mathrm{g/kg}$ liquid water related to total number of cells at that level, and an approximated air density of $1\,\mathrm{kg/m}^3$, this leads to a medium cloud droplet density of $20\,\mathrm{mg}^{-1}$ which is in good agreement with the work of Slawinska et al. [2011]. The liquid water path and the turbulent kinetic energy show a very slight increase over the complete 16 hours simulation time while the cloud fraction remains constant. The liquid water path reaches values between $10\,\mathrm{g/m}^2$ and $15\,\mathrm{g/m}^2$. The cloud cover ranges from 0.1 up to 0.2. All values are in good agreement with the results of Siebesma et al. [2003] (figure 19) although the liquid water path is a bit overestimated. The resolved turbulent kinetic energy is with $250\,\mathrm{kgm}^2/\mathrm{s}^2$ slightly lower compared to about $400\,\mathrm{kgm}^2/\mathrm{s}^2$ at Siebesma et al. [2003]. This is also recognizeable in the comparison of the vertical profiles of the turbulent kinetic energy in figures 20 and 21. One possible reason for this is the quite diffusive third order upwind scheme for advection which dissipates higher frequency oscillations and eddies. The strongest deviation is located near the surface, where the TKE from Siebesma et al. [2003] shows values up to $0.4\,\mathrm{m}^2/\mathrm{s}^2$ while the maximum TKE value in the model ASAMgpu is at $0.25\,\mathrm{m}^2/\mathrm{s}^2$ (see figure 20 and 21). Above the near surface maximum both models show a decrease of TKE above 500 m and a second peak at 1500 m. Again the second peak in ASAMgpu is not as strong as the median from Siebesma et al. [2003]. Also the distribution of the liquid water content with height differs from the ensemble mean. The model ASAMgpu produces a stronger peak with $0.010\,\mathrm{g/kg}$ at a height of about 900 m and then decreases to $0.003\,\mathrm{g/m}^3$ at 1600 m. This peak can not be found in the profile from Siebesma et al. [2003], where a more homogeneous distribution of the liquid water content with height is presented. The work of Slawinska et al. [2011] showed similar vertical profiles for cloud

fraction and cloud droplet density like the ASAMgpu model, if in cloud activation of cloud condensation nuclei is suppressed. The referred work uses a double moment warm microphysical scheme by Morrison and Grabowski [2008]. The profiles in Slawinska et al. [2011] change to the more isotropic behaviour with enabled in cloud droplet activation. The background density of potential cloud condensation nuclei for the ASAMgpu runs is initialized with $100\,\mathrm{mg}^{-1}$. This leads to a maximum of domain averaged cloud droplet density of $1.5 \cdot 10^6/\mathrm{m}^3$. Together with a cloud fraction of 0.08, which is the number of cells with more than 0.01 g/kg liquid water related to total number of cells at that level, and an approximated air density of $1\,\mathrm{kg/m}^3$, this leads to a medium cloud droplet density of $20\,\mathrm{mg}^{-1}$ which is in good agreement with the work of Slawinska et al. [2011].

Figure 23 shows the mean power spectrum in space at a height of 150 m integrated over the last 2 hours of the simulation. To achieve this spectra 256 horizontal lines were sampled and analyzed using a Fast Fourier Transformation (FFT). The showed spectrum is the average of the spectra for every single horizontal line and for every time step during the last two hours. The dotted line represents the theoretical decay of the spectral density in the inertial subrange of the spectra. The model reproduces the dissipation for larger eddies and increases dissipation in the higher frequencies. Especially in the direction parallel to the mean wind velocity (X) the $-5/3$ decay is simulated quite well, while in the perpendicular direction (Y), higher energy densities can be found at wavenumbers between $10^{-3}\,\mathrm{m}^{-1}$ and $20^{-2}\,\mathrm{m}^{-1}$. Those wavenumbers correspond to convective rolls evolving parallel to the wind velocity.

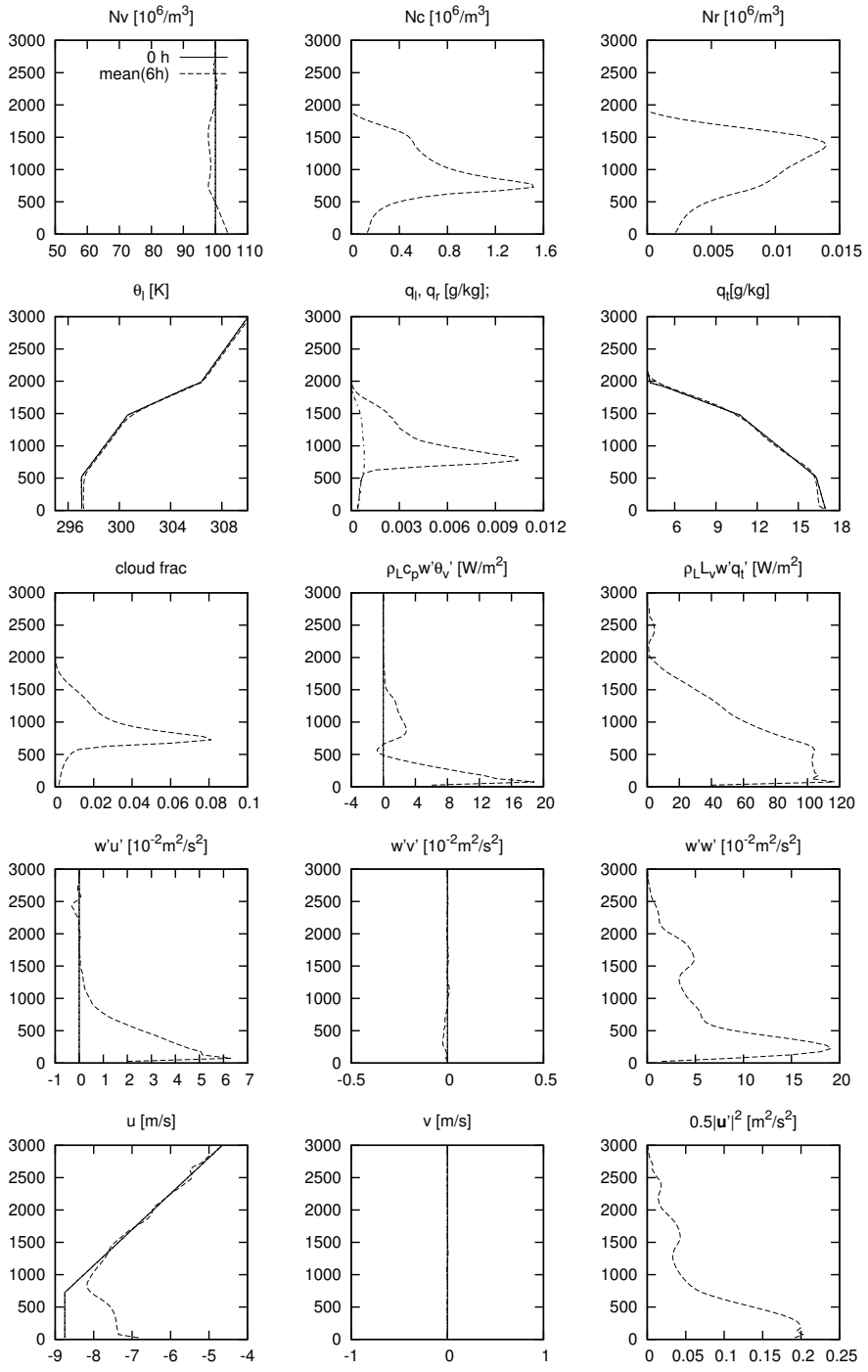

Figure 20 Vertical profiles for the BOMEX test case at the beginning and averaged over the 6th hour.

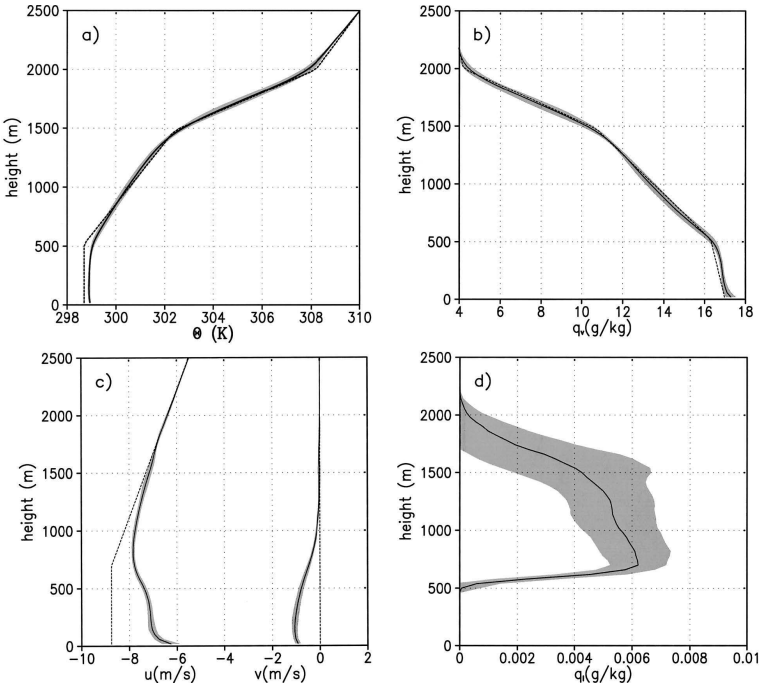

Figure 21 Mean profiles for the BOMEX case from Siebesma et al. [2003] averaged over the 6th hour of (a) potential temperature, (b) water vapor specific humidity, (c) the horizontal velocity components, and (d) the liquid water q_l. The solid lines indicate the average and the band is a width of twice the standard deviation of the models participated in the comparison. The dashed lines indicate the initial profiles.

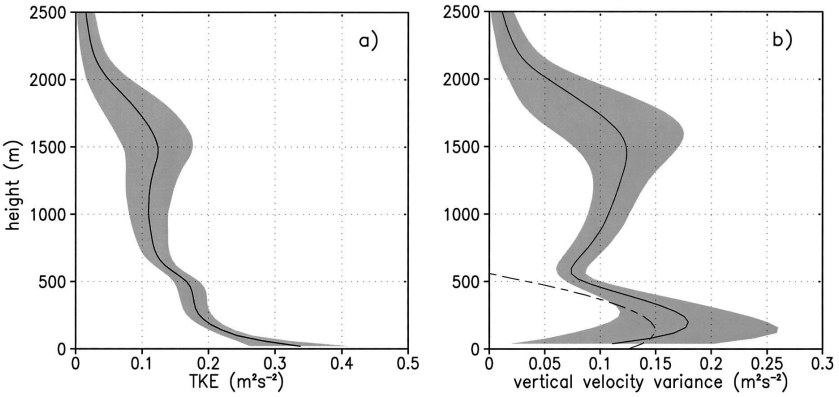

Figure 22 Vertical profiles from Siebesma et al. [2003] of (a) turbulent kinetic energy (TKE) and its vertical component (b) σ_w^2. The dashed line in (b) corresponds to a mixed-layer relationship for a dry boundary layer (see Siebesma et al. [2003] for details).

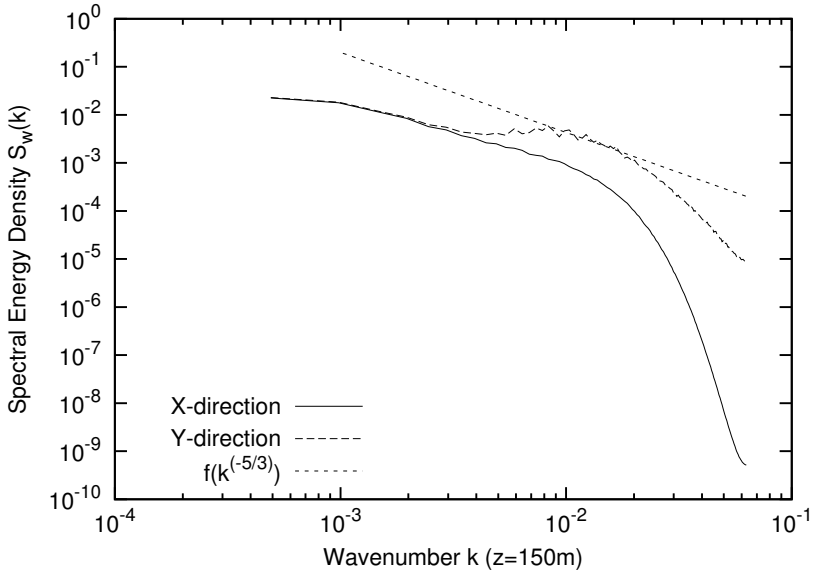

Figure 23 Turbulence power spectra for the BOMEX test case averaged over the last 2 hours in X- and Y-direction. The function f(x) represents the theoretical $-5/3$ decay in the inertial range.

4.6 Non-drizzling Stratocumulus: DYCOMS-II research flight one (RF01)

This section presents the results for a simulation based on measurements from the research flight one (RF01) during the second Dynamics and Chemistry of Marine Stratocumulus field study (DYCOMS-II) that took place over the Pacific Ocean near the coast of California. The setup follows Stevens et al. [2005] trying to reproduce a nondrizzling boundary layer structure measured during the flight. All initialisation profiles and surface fluxes for this example are applied like described in Stevens et al. [2005]. The main difference to the BOMEX case from the last section is the two layer setup with a well mixed boundary layer topped by a very strong inversion. The calculations presented in this section were performed in a domain with $256 \times 256 \times 32$ grid cells and an isotropic resolution of 50 m. The initial moisture and temperature profiles are defined as follows.

$$
\begin{aligned}
& & & q_t[\mathrm{g/kg}] = \\
0 < & z & <840 & \quad 9.0 \\
840 < & z & & \quad 1.5
\end{aligned}
\tag{53}
$$

$$
\begin{aligned}
& & & \theta_l[\mathrm{K}] = \\
0 < & z & <840 & \quad 289.5 \\
840 < & z & & \quad 297.5 + \qquad (z - 840)^{1/3}
\end{aligned}
\tag{54}
$$

With this conditions a cloud already exists in the initial state. The density of available CCN's in the cloud is initialized with a constant value of $65\,\mathrm{cm}^{-3}$. The initialized droplet number density is not specified in the original work, so a simple proportionality between cloud water content and droplet number density is assumed.

$$N_c[1/\mathrm{m}^3] = \frac{\rho_l}{8.47 \times 10^{-4}} \cdot 55 \times 10^6 \tag{55}$$

With that assumption a maximum cloud droplet density of $33\,\mathrm{cm}^{-3}$ at a liquid water content of $0.45\,\mathrm{g/m}^{-3}$ was reached. To reduce numerical diffusion the whole domain is subject to a Gallileian transformation with the geostrophic wind of $U = 7\,\mathrm{m/s}$ and $V = -5.5\,\mathrm{m/s}$. Surface fluxes are constant $115\,\mathrm{W/m}^{-2}$ for the latent and $15\,\mathrm{W/m}^{-2}$ for the sensible heat flux. A large scale subsidence is defined with $W = -3.75 \times 10^{-6}\,\mathrm{s}^{-1} \cdot z$. In addition a simple long-wave radiative forcing is parameterized in dependence of the liquid wather path in the column above and below the current position following Stevens et al. [2005].

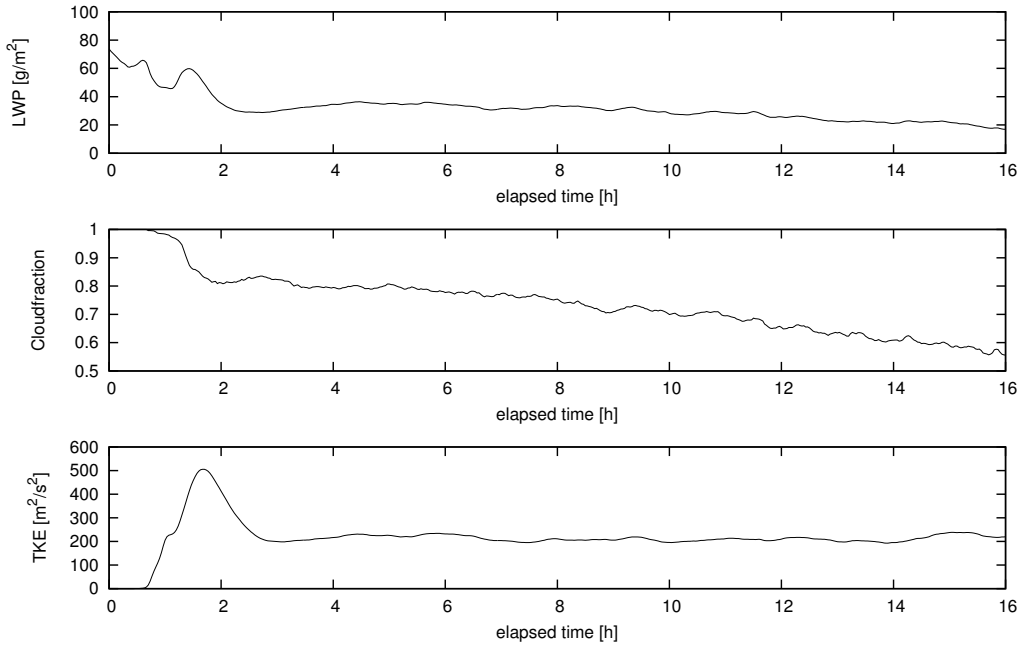

Figure 24 Time series for the liquid water path (LWP), cloud fraction and the vertically integrated turbulent kinetic energy for the DYCOMS-II RF01 case.

The time series for this example show a spin up phase for the first three hours. During this phase the vertical motions in the boundary layer develop

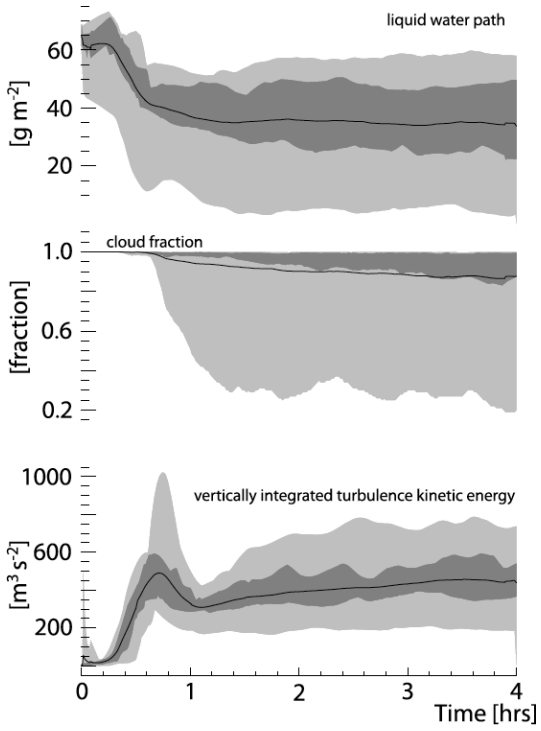

Figure 25 Time evolution of the first 6 hours of the DYCOMS II RF01 test case from Stevens et al. [2005].

and a peak in the turbulent kinetic energy is observable. The vertical motions lead to entrainment of warmer dry air at the inversion layer. Liquid water path decreases from initial $80\,\mathrm{g/m^2}$ down to $30\,\mathrm{g/m^2}$. At the same time the cloud fraction decreases from the initialised 1 down to 0.7. Like in the BOMEX case the spin up phase in ASAMgpu took about as twice as long as the model mean presented in Stevens et al. [2005], most probably again through a stronger concentration of potential energy through the missing subgrid scale turbulence. After the spin up phase the system is near the equilibrium with the driving fluxes and a stady state evolves. Because the low vertical resolution leads to an overestimated entrainment at the inversion, cloud cover still reduces slightly.

The vertical profiles averaged over the fourth hour of the simulation (figure 26) are in good agreement with the results from the inter comparison (figure 27). Temperature and moisture profiles show a constant behavior

through the well mixed layer below the cloud. The initial strong gradients in the profiles of moisture and temperature near the inversion are reduced. Most probably the advection scheme, which tends to produce oscillations in the presence of sharp gradients, and the enhanced spin up phase are the reasons for the mixing of warm and dry air from above the inversion into the cloud layer. The maximum liquid water content reaches $0.2\,\mathrm{g/m^3}$ at a height of 750 m. With that the maximum is slightly lower and at a reduced height compared to the master ensemble in Stevens et al. [2005] (figure 27) but still in the range of the participated models.

Furthermore the vertical mean profiles of the variance of the vertical velocity and the turbulent kinetic energy show a tendency to develop a decoupled cloud layer. This can be observed through the small peak in the TKE and the change in gradient in the vertical velocity variance at cloud level. After Stevens et al. [2005] this may happen if the simulation tend to produce a warmer state with less cloud water.

The turbulence spectra (figure 28) show no destinctive features, but it is recognizeable that more turbulent kinetic energy is present in the Y-direction as a product of the higher mean wind speed in this direction. The simulation shows that the GPU model is able to reproduce main features of the DYCOMS-II RF01 case. The differences to the studies in Stevens et al. [2005] can be explained mainly through the third order upwind advection scheme especially in such a thin cloud layer with a thickness of 200 m (four grid cells) and the very sharp gradient at the inversion.

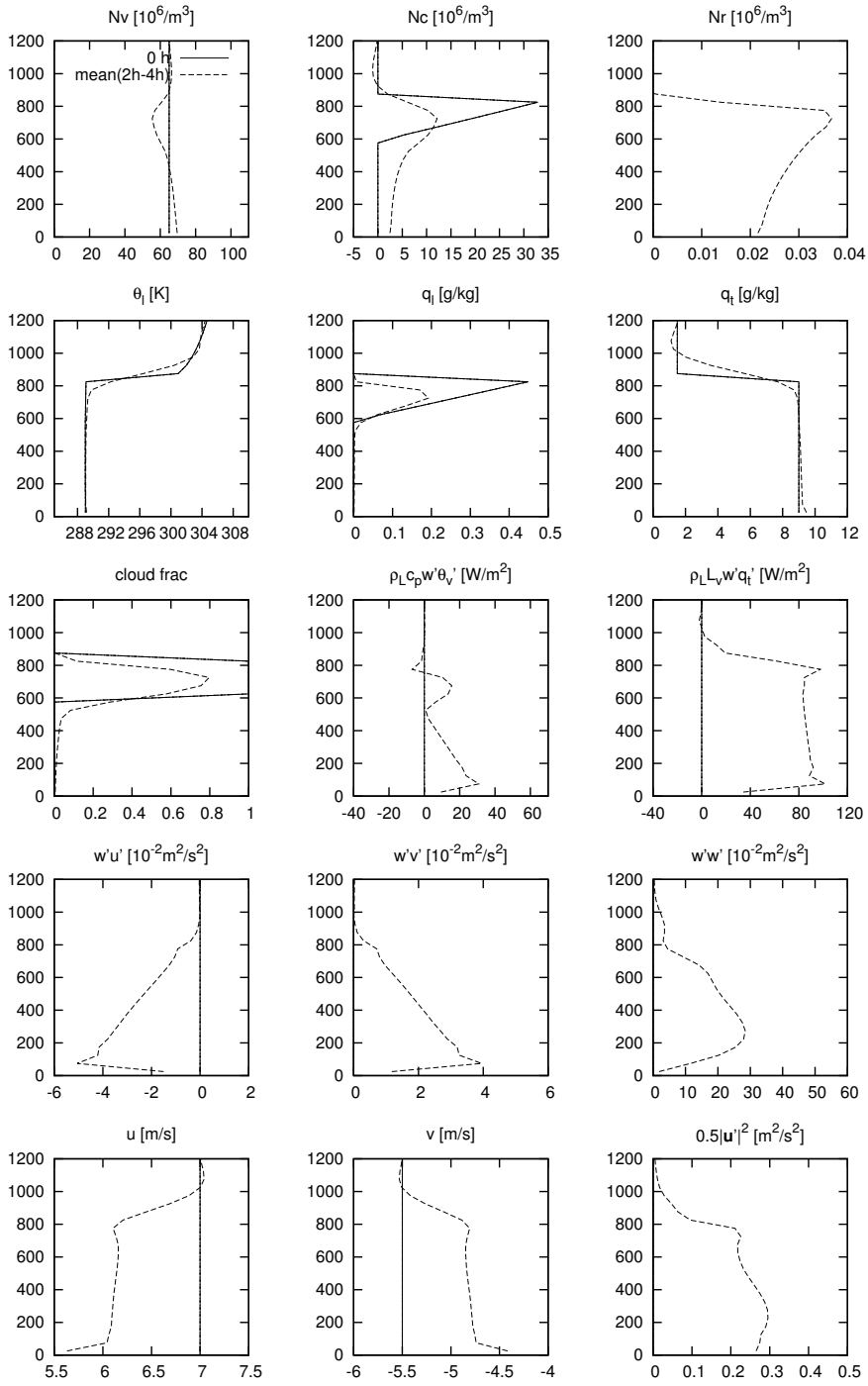

Figure 26 Mean vertical profiles for the DYCOMS-II RF01 test case for the fourth hour.

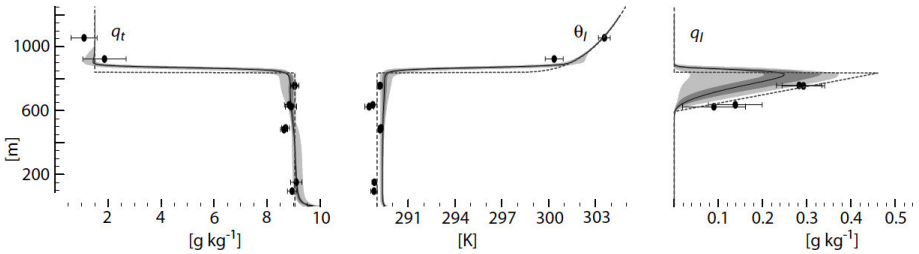

Figure 27 Vertical profiles from Stevens et al. [2005] showing the initial state (dashed), observations (dots) and values averaged over the 4th hour of the DYCOMS-II RF01 test case.

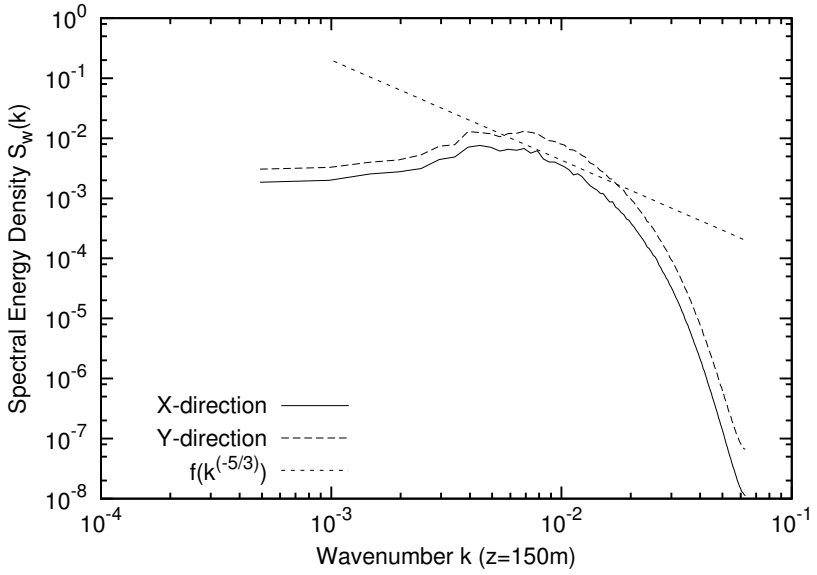

Figure 28 Turbulence power spectra for the DYCOMS-II RF01 test case averaged over the last 2 hours in X- and Y-direction. The function f(x) represents the theoretical -5/3 decay in the inertial range.

4.7 Rain in Cumulus over the Ocean: RICO

The third test case is based on the Rain in (shallow) Cumulus over the Ocean Campaign (RICO). The campaign was focused on the development and evolution of precipitation in cumulus clouds. It took place in the western Atlantic in the time from November 2004 to January 2005. Detailed information about the campaign and the performed measurements can be found in

Rauber et al. [2007]. The surface fluxes were implemented after VanZanten et al. [2011] as a function of wind speed in the first model layer using

$$\begin{aligned}
\overline{w'\theta'} &= -C_h||U||(\theta - \theta|_{z=0}) \\
\overline{w'q_t'} &= -C_q||U||(q_t - q_{sat}|_{z=0}) \\
\overline{u'w'} &= -C_m||U||u \\
\overline{v'w'} &= -C_m||U||v
\end{aligned} \tag{56}$$

with $C_h = 0.001094$, $C_q = 0.001133$ and $C_m = 0.001229$. This flux definition using the windspeed in the first level may produce differences between models using an explicit subgrid turbulence scheme and models without. Such a scheme would reduce the vertical wind shear induced by the frictional forces at the surface through enhanced momentum diffusion. In a model without a subgrid turbulence scheme the horizontal wind speeds in the first level are reduced leading to smaller surface fluxes. The initial profiles for moisture and temperature are defined as follows.

$$q_t[\mathrm{g/kg}] = \tag{57}$$

$0 <$	z	<740	$16.0 + (13.8 - 16.0)/(740)$	$\cdot z$
$740 <$	z	<3260	$13.8 + (\ 2.4 - 13.8)/(3260 - 740)$	$\cdot(z - 740)$
$3260 <$	z	<4000	$\ 2.4 + (\ 1.8 - \ 2.4)/(4000 - 3260)$	$\cdot(z - 3260)$

$$\theta_l[\mathrm{K}] = \tag{58}$$

$0 <$	z	<740	297.9	
$740 <$	z	<4000	$297.9 + (317.0 - 297.9)/(4000 - 740)$	$\cdot(z - 740)$

In those profiles no clouds are present. The background cloud condensation nuclei (CCN) population density in the original work was fixed at $70\,\mathrm{cm}^{-1}$. For models that predict the cloud-droplet activation spectrum this value was assumed to be fixed at $100\,\mathrm{cm}^{-1}$ at 1% supersaturation. In con-

trast to that, the ASAMgpu model contains a prognostic variable for the background aerosol that serve as possible CCN's. This is not the aerosol density but the density of the aerosol fraction activated at a supersaturation of 1%. The number of finally activated CCN's is a function of supersaturation, the mentioned CCN density variable and the product of the vertical gradient of supersaturation and the vertical velocity. This available CCN density was initialised at $100\,\mathrm{cm}^{-1}$ as well and reduces during simulation time down to $80\,\mathrm{cm}^{-1}$ through coagulation and scavenging by drizzle. Further applied large scale forcings are a subsidence, a radiative cooling of $2.5\,\mathrm{K\,day}^{-1}$, and a large scale advection term, which emulates the transport of dry air into lower layers and moist air into upper ones, defined as follows.

$$
\begin{aligned}
& & & w_{subsidence}[\mathrm{m/s}] = \\
0 < & z & <2260 & \quad -(0.005/2260)\cdot z \\
2260 < & z & <4000 & \quad -0.005
\end{aligned}
\tag{59}
$$

$$
\begin{aligned}
& & & d\theta/dt[\mathrm{K/s}] = \\
0 < & z & <4000 & \quad -2.89\cdot 10^{-5}
\end{aligned}
\tag{60}
$$

$$
\begin{aligned}
& & & dq_t/dt[\mathrm{g/kg\,s}^{-1}] = \\
0 < & z & <2980 & \quad -0.116\cdot 10^{-6} + 4.116\cdot 10^{-6}/(2980)\cdot z \\
2980 < & z & <4000 & \quad 4\cdot 10^{-6}
\end{aligned}
\tag{61}
$$

The domain size for this simulation differs from the original work and was choosen with $256 \times 256 \times 64$ cells at a resolution of 60 m in the horizontal and vertical direction. For time integration an advection time step of 1 s and a fast time step of 1/18 s for pressure waves and microphysics were used. Overall simulation time was 24 hours.

Just as in the previous two cases the boundary layer is initialized in complete rest pertubated by very small random fluctuations in θ_l and the water vapor content ρ_v. This leads to a strong spin up phase until the

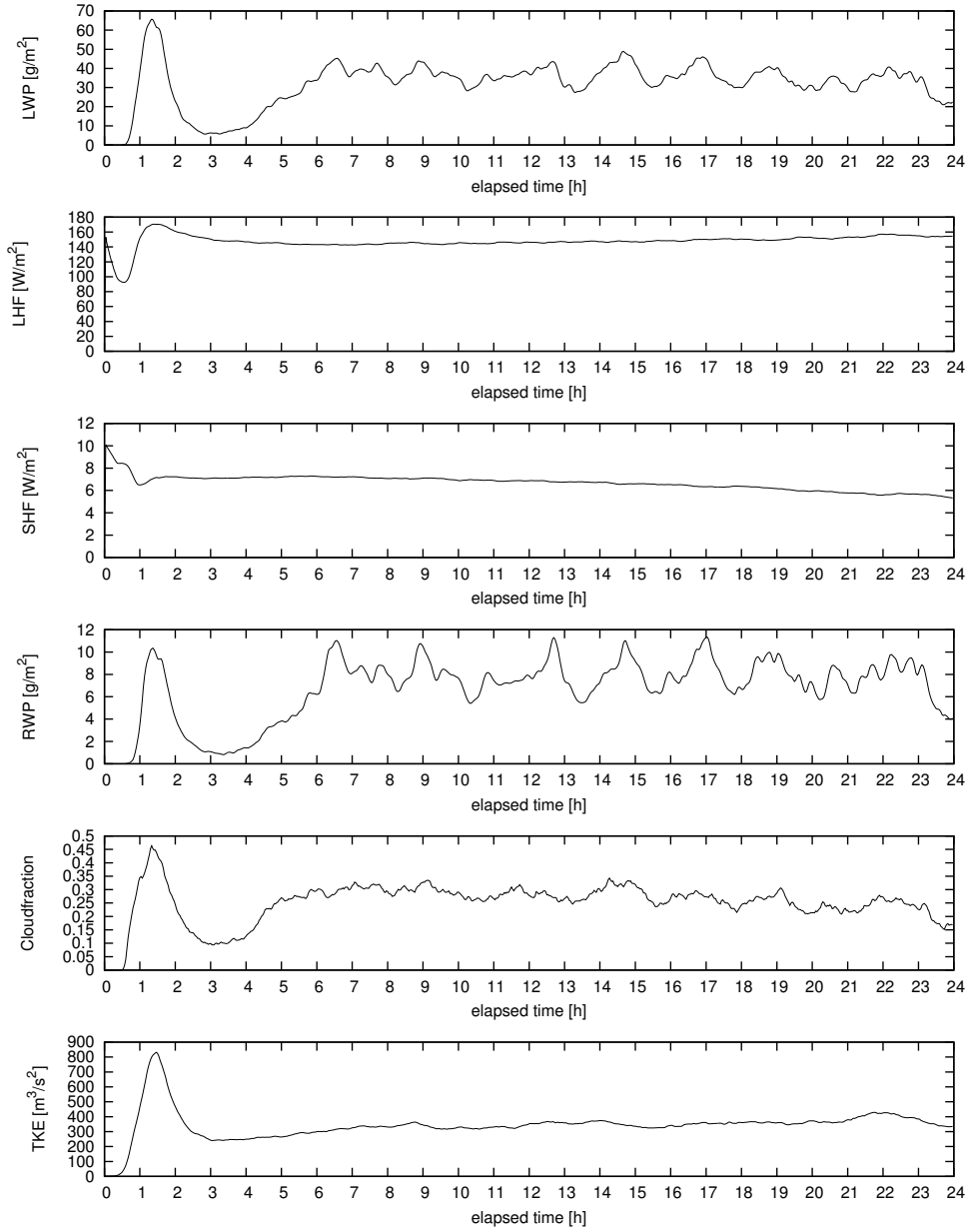

Figure 29 Time series for the RICO test case from the model ASAMgpu.

convective structures in the boundary layer developed. During this spin up a lot of cloud water and in this case also drizzle is produced. After that a short phase with less clouds occurs, followed by small cumulus clouds growing

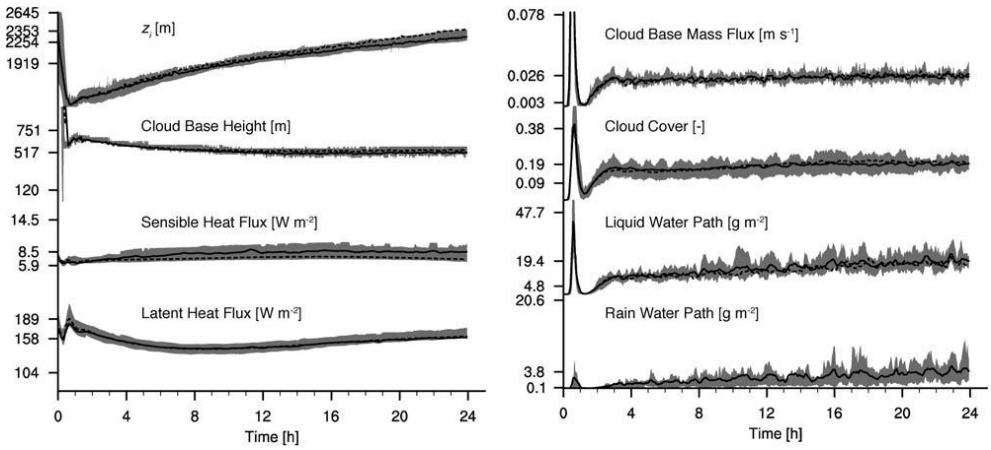

Figure 30 Times eries for the RICO test case from VanZanten et al. [2011].

to larger drizzling cumulus clouds ascending in the cloud layer against the inversion up to a height of 1900 m. Long term evolution of this example shows constant surface fluxes at 160 W/m^2 for the latent and 7 W/m^2 for the sensible heat flux. The liquid water path is too high for the complete simulation time. The reason for that could not yet be identified. Experiments showed a reduction of liquid water path with deactivated nucleation, which effectivly reduces the microphysics to a single moment scheme. The high rain water path could be a hint that the reason may be connected to the autoconversion process, but it even could be some numerical issue concerning the single precission during evaluation of the microphysical forcings. At this point further evaluation is still necessary.

Vertical profiles averaged over the last 4 hours (figure 32), show a maximum in the liquid water content at 1250 m with 50 mg/m^3 which is too high compared to the results in VanZanten et al. [2011]. In the cited work it is also mentioned that results especially for the maximum liquid water content and cloud fraction differ strongly between the different models. In addition experiments with the UCLA-LES model (Matheou et al. [2011],Nuijens [2010]) using different advection schemes, time stepping methods or even mean wind speeds produce commensurate or even larger differences in the results than between the models presented in the cited work. The main reason for that

is that the strength and the height of the trade-inversion is very sensitive to numerical formulations because it has to develop in the model more or less spontaneously, in contrast to the other cases where it was initialized with a very strong gradient in the temperature and moisture profiles.

Contrary to the other two examples, where the power spectra were aquired near the surface, the spectra for this case (figure 33) were calculated near the cloud base at a height of 700 m to ensure comparability to Matheou et al. [2011]. The results for the spectrum are in good agreement with the cited work.

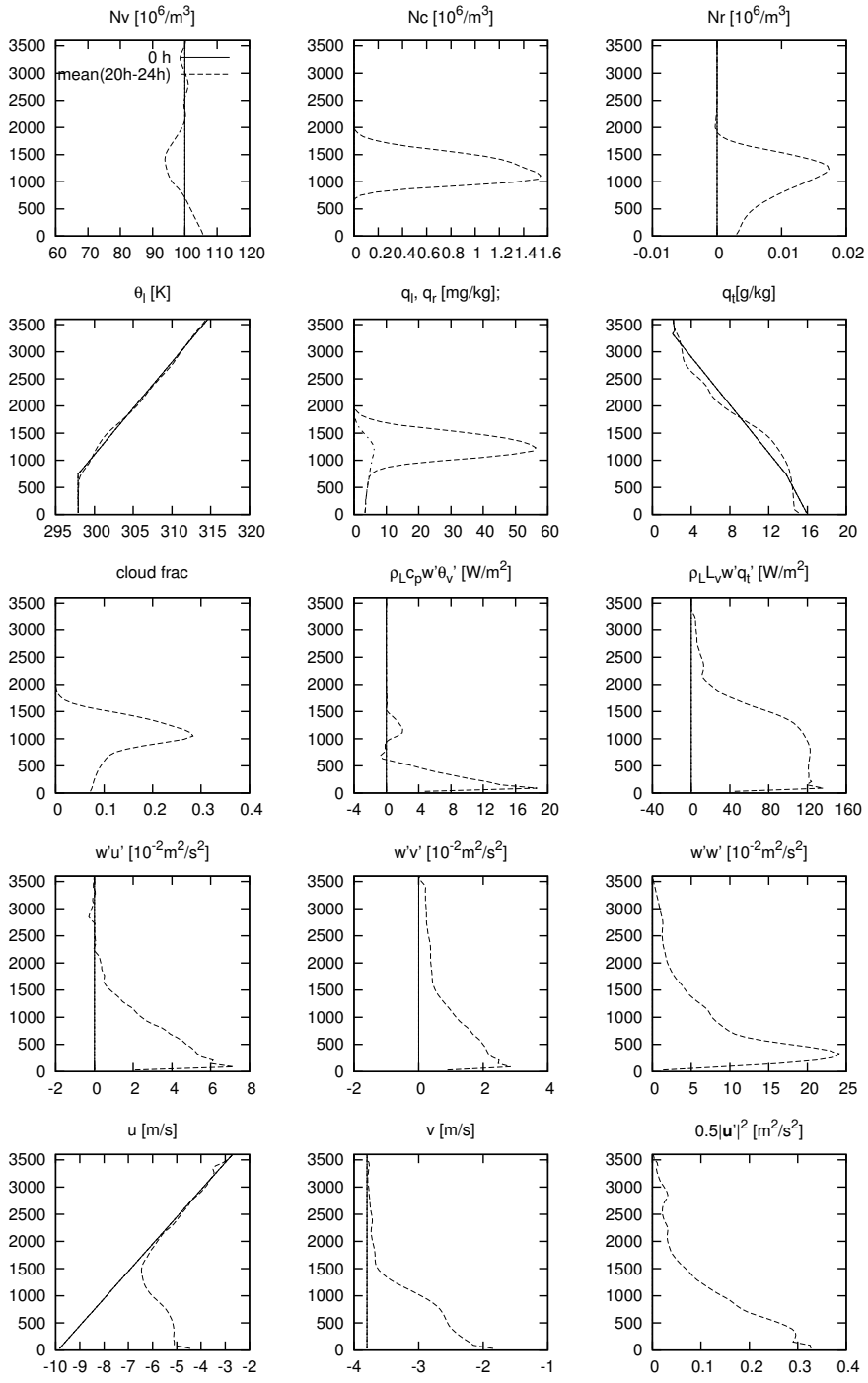

Figure 31 Vertical profiles for the RICO test case at the beginning and mean profile averaged over the hours 20-24.

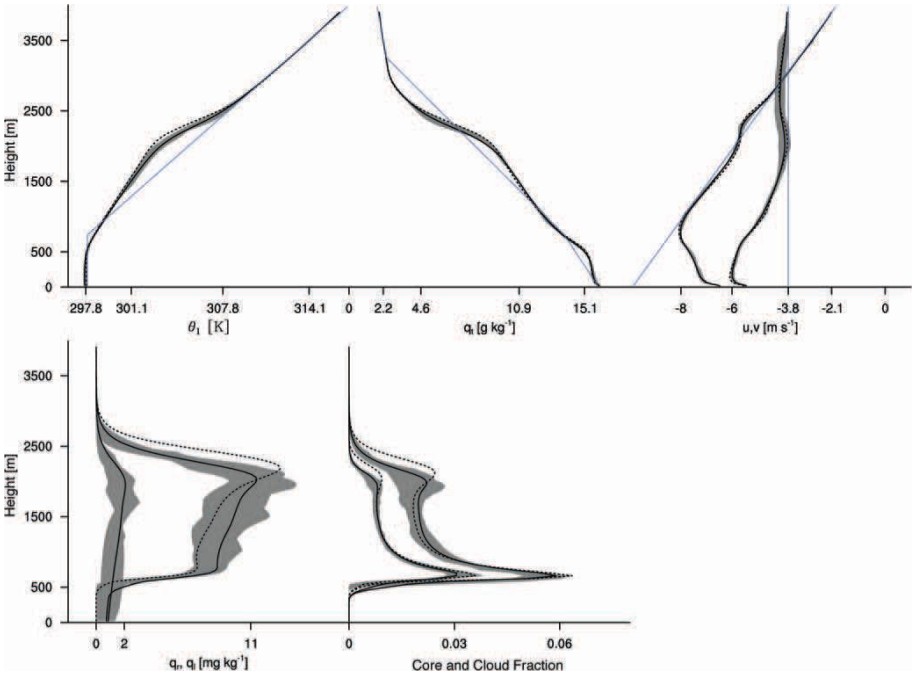

Figure 32 Vertical profiles from VanZanten et al. [2011] for the RICO test case at the beginning and mean profile averaged over the hours 20-24.

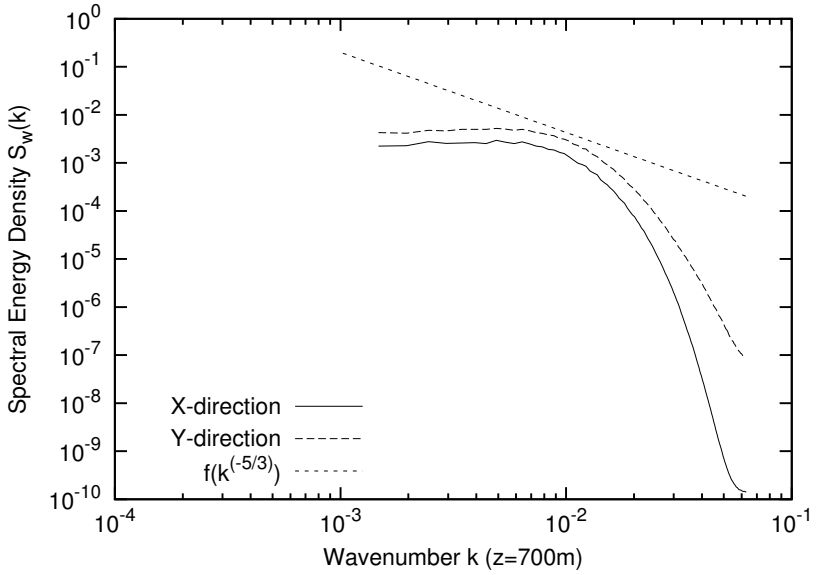

Figure 33 Turbulence power spectra for the RICO test case averaged over the last 2 hours in X- and Y-direction. The function f(x) represents the theoretical $-5/3$ decay in the inertial range.

4.8 Real Case: Kap Verde Islands

This section describes the simulation of the effect of a flat heat island on the turbulence in the marine boundary layer. Most of the information were already published as a part of Engelmann et al. [2011]. It was motivated by Doppler lidar measurements during the SAMUM-2 campaign in the year 2008. The Doppler lidar was located in the southeast of the Cap Verde island Santiago at the Airport of the capital city Praia (Ansmann et al. [2011]). Figure 34 shows the vertical velocity measurement above the super site at the 25th of January, 2008. At a height between 600 m and 1100 m a relatively consistent vertical updraft with a velocity of about 0.5 m/s topped up to a height of 2000 m by a vertical downward motion of the same order can be observed.

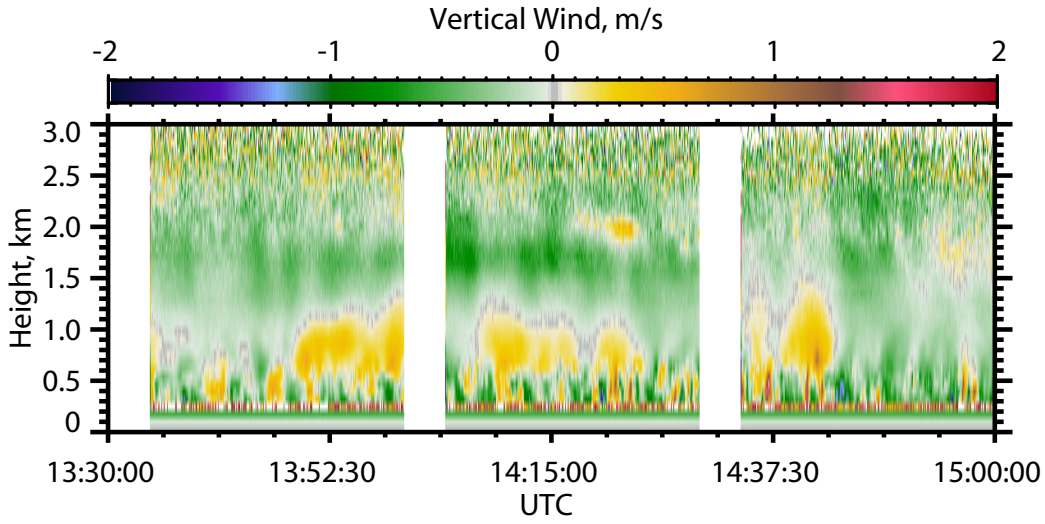

Figure 34 Vertical velocity above supersite at Cap Verde Island during SAMUM-II Campaign measured with the wind LIDAR Willy (from Engelmann et al. [2011]).

In addition the Falcon research aircraft of the DLR provided lidar measurements of the vertical distribution of aerosol layers in and above the marine boundary layer.

For a better interpretation of those measurements a simple large eddy simulation setup was used. The two islands Santiago and Maio were modeled

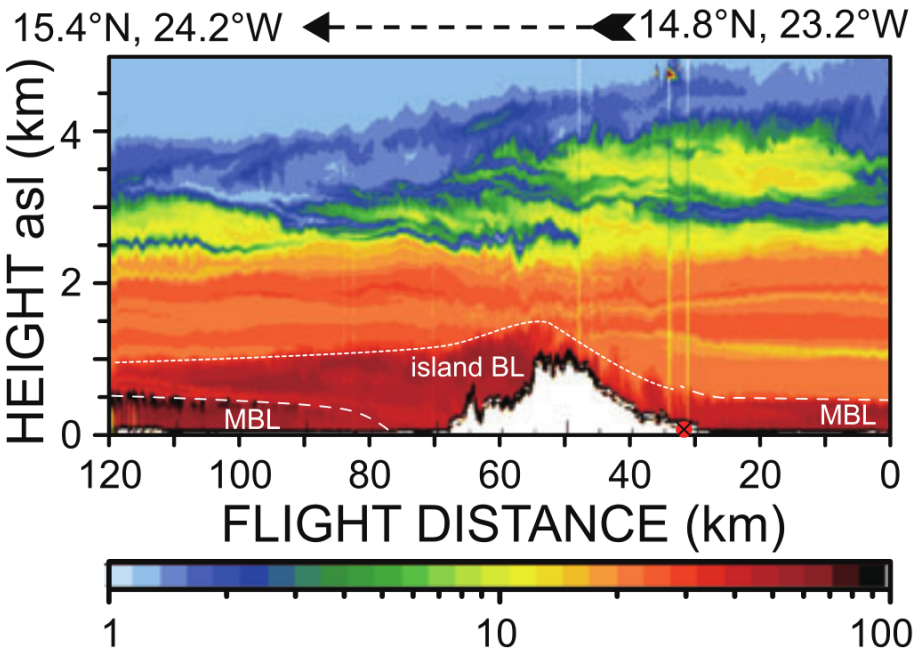

Figure 35 Aerosol backscatter ratio (total/Rayleigh backscatter) at 1064 nm measured with HSRL aboard the DLR Falcon research aircraft on 25th of January. The aircraft crossed the ground site (red circle at distance 32 km at approximately 15:30 UTC). The flight course is shown in figure 40 (Engelmann et al. [2011]).

as flat heating surfaces in the ocean. The overall domain size was $256 \times 256 \times 62$ at a horizontal resolution of 360 m and a vertical resolution of 60 m. This results in a simulation area of $92 \times 92\,\mathrm{km}^2$ and a simulation domain height of 3.7 km. The advective time step for this simulation was at 1 s and the acoustic at 1/12 s. Because of missing values for the sensible and latent heat fluxes for the soil model, data of the Global Data Assimilation System (GDAS) of the United States National Centers for Environmental Prediction were used. Since the islands are not completely resolved by GDAS, the data were approximated using the nearby continental area in the outbacks of Dakar, Senegal. The surface characteristic is comparable. The sensible heat flux above the islands was modeled using a diurnal cycle with a minimum of $-77\,\mathrm{W/m}^2$ and a maximum at $516\,\mathrm{W/m}^2$, using the following function (62).

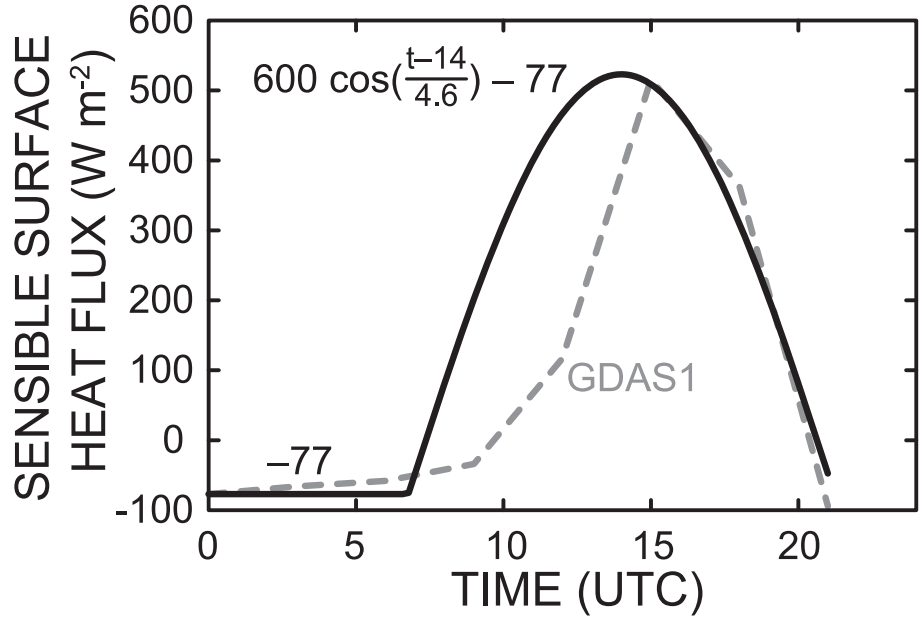

Figure 36 Diurnal parameterization of the sensible surface heat flux for the Cap Verde simulations (from Engelmann et al. [2011]).

$$F_{surf/day}[\mathrm{W/m^2}] = -77 + 600 \cdot \cos\left(\left(t_{local[h]} - 14\right)/4.6\right) \qquad (62)$$
$$F_{surf/night}[\mathrm{W/m^2}] = -77$$

The latent heat flux was constant at $55\,\mathrm{W/m^2}$. Figure 36 shows the parameterized sensible surface flux compared to GDAS data. Because of the much lower vertical resolution the GDAS data contain a parameterized boundary layer, which causes the exponential growth in the surface heat flux in the morning. This boundary layer can be resolved in high resolution simulations. The assumption that the surface heat flux is proportional to the incoming radiation leads to a simple cosine function with a maximum similar to the GDAS data. During night the islands are cooled by radiation, approximated by a negative sensible heat flux of $77\,\mathrm{W/m^2}$.

Above the ocean constant marine surface fluxes of $90\,\mathrm{W/m^2}$ latent heat flux and about $20\,\mathrm{W/m^2}$ sensible heat flux were applied. To break the sym-

metry, the latent heat flux is perturbed by a random noise of $5\,\mathrm{W/m^2}$. One problem were the lateral boundary conditions at the in- and outflow boundaries. To provide a consistent turbulent flow field, including vertical velocities and turbulent boundary layer structures like convective cells, a second simulation was performed in parallel. This second simulation was processed in a domain without the island, using periodic boundary conditions and initialized with the sounding of the simulated day. In this domain an undisturbed marine boundary layer was able to evolve, just controlled by the initial state and the surface heat fluxes. Now three cells from the boundary of this period domain were used as nesting boundary conditions for the domain containing the islands. With this approach a more or less realistic inflow for the island domain could be realized without the need for a very long forerun range in the upwind region. All model runs started at 8:00 local time and the data shown is from 15:30.

In figure 37 and figure 39 horizontal cutplanes of the model output for the vertical velocity fields at the 23th and 25th of January, 2008 are presented. On both days the model simulations show a strong tendency to produce smaller at the coast induced updrafts, which perform a self organization and converge to one strong updraft region above the heated island surface. This recurring updraft region above the island is presumably the basic structure for forming cloud streets behind such an island, although in the two presented test cases, moisture was not sufficient to produce clouds. In addition, to illustrate mixing processes induced by the heat island, the vertical distribution of a passive tracer initialized in the marine boundary layer is shown.

At the 23th of January vertical velocities of the convective structures in the boundary layer were in the range of $-4\,\mathrm{m/s}$ to $4\,\mathrm{m/s}$. The already mentioned induced updrafts finally produce one stronger updraft above the island. During this day this larger updraft reached wind speeds up to $9.5\,\mathrm{m/s}$. The thick black line in figure 37 indicates the position of the vertical cut plane where the vertical distribution of the simulated passive tracer as a representation of the marine boundary layer aerosol is presented. The initial step profile of the tracer at the domain boundaries was 1 up to 700 m height

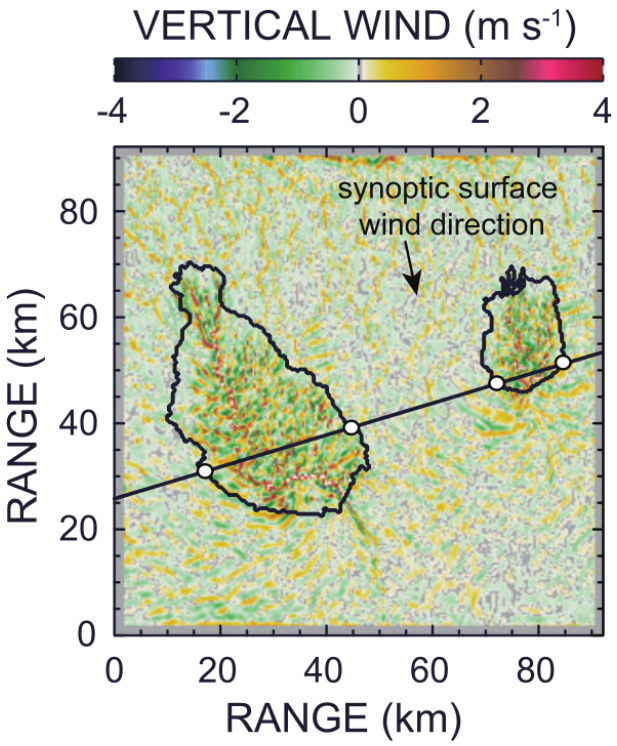

Figure 37 Vertical wind component over Santiago and Maio derived for a height of 400 m above sea level and at 15:30 UTC on 23 January 2008. For the solid line with circles (indicating the island boundaries) height-distance profiles are shown in figure 38 (from Engelmann et al. [2011]).

and 0 for heights above 700 m. During this day the passive tracer reached a height of 1500 m above the islands, even above the comparably small island of Maio. Also in the area between 45 km and 55 km a lofted aerosol layer can be found in the model data.

In contrast to the 23th, the prevalent vertical velocities at the 25th of January were a bit lower in the range of -2 m/s to 2 m/s, except in the organized updraft above the island where it even reached higher velocities up to 11 m/s. Another very interesting feature in this flow field is the sea breeze structure in front of the north easterly coastline of the island of Santiago, showing very consistent up- and downward motions also above the measurement site at Praia in a velocity range of ± 0.5 m/s. This could be one possible explanation for the structures that were found in the lidar measurements above the

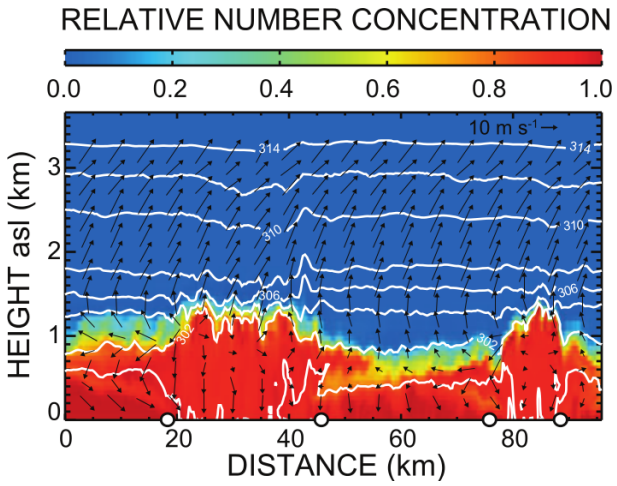

Figure 38 Relative number concentration of passive boundary-layer aerosol tracer, isentropic levels and horizontal wind speed and direction (arrows) at the cut plane indicated in figure 37 for the 23th of January 2008 at 15:30 UTC. The islands of Santiago and Maio are located from $18 - 46$ km and from $76 - 88$ km, respectively (indicated by open circles) (from Engelmann et al. [2011]).

boundary layer in a heigt from 800 m to 1200 m. During this day also the Falcon measurement took place. The vertical aerosol distribution plot for this day is reoriented to be parallel to the flight path of the Falcon airplane, indicated by the solid and dotted black lines in figure 39. The comparison between the model output (figure 40) and the Falcon measurements (figure 35) concerning boundary layer depth show a good agreement. The boundary layer in front of the island is mainly the original marine boundary layer up to a height of 800 m. Above the island the Falcon measured a maximum boundary layer height of 1500 m and behind the island in the flight path the boundary layer height was determined with slightly below 1000 m. Even with this simple flat heated planar surface as an island, all values are in very good agreement with the model results.

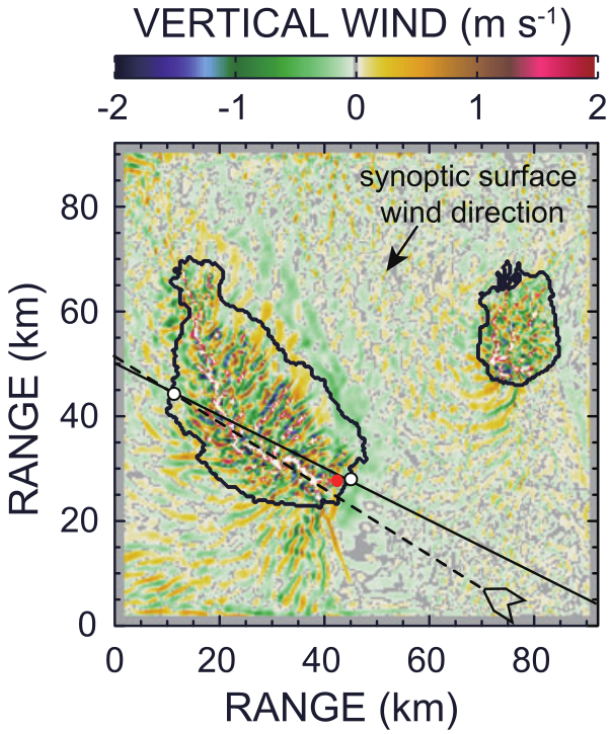

Figure 39 Vertical wind component over Santiago and Maio derived for a height of 400 m above sea level and at 15:30 UTC on the 25th of January 2008. For the solid line with circles (indicating the island boundaries) height-distance profiles are shown in figure 40 (from Engelmann et al. [2011]).

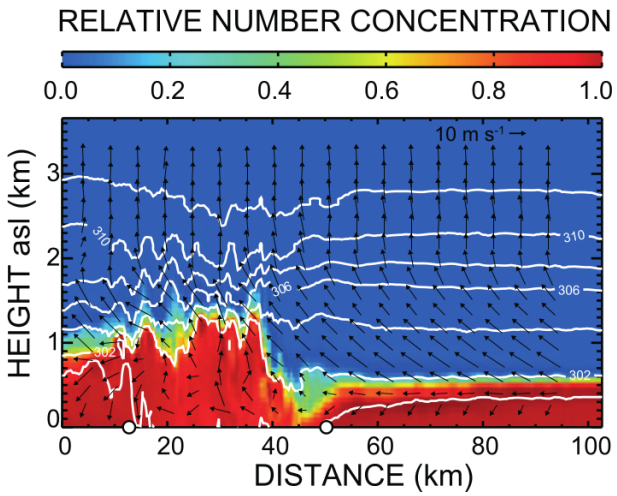

Figure 40 Relative number concentration of passive boundary-layer aerosol tracer, isentropic levels and horizontal wind speed and direction (arrows) at the cut plane indicated in figure 39 for the 25th of January 2008 ats 1530 UTC. The islands of Santiago and Maio are located from $18-46$ km and from $76-88$ km, respectively (indicated by open circles) (from Engelmann et al. [2011]).

5 Summary

The presented work evaluates the possibility to use the general purpose computation on graphic processing units (GPGPUs) with the example of a three dimensional atmospheric model (ASAMgpu). Instead of the widely used proprietary CUDA interface, the presented approach relies on open source and object oriented techniques. The main program is written in C++ using OpenGL and the OpenGL Shader Language (GLSL) as a reliable interface for GPU access. Two classes were developed to encapsulate the functions necessary for texture and shader handling. The resulting framework provides a easy to use interface for data flow control and shader handling.

Overall the new model ASAMgpu allows to perform high resolution state of the art atmospheric simulations on inexpensive hardware and with low power consumption in time ranges prior only possible on large clusters of supercomputers. Simulations with a typical domain size of $256 \times 256 \times 64$ cells at a resolution of 60 m using a timestep of 1 s with 18 acoustic steps and a simulation time of 24 hours could be realized within 16 hours computation time on one single GPU. Future development will include a transition to OpenCL which should be straight forward, because the structure of the source code is compatible. Also the development on the GPU market will proceed, so strong accelerations can be expected there. The restriction to single precission demands caution in numerical formulations, and may be one reason for some still unsolved problems. Especially in the RICO case, with larger amount of drizzle included, liquid water path and rain water path could not be reproduced to complete satisfaction.

The implicit dissipation in the advection scheme is able to reproduce the $-5/3$ decay in the power spectra during the simulations. Also the model formulation using the new thermodynamic variable derived in this work produces reasonable results and simplifies the implementation of microphysical parameterizations, hence sources only occure during phase transitions or in case of external fluxes (e.g. surface fluxes). Because those sources consist just of a few simple terms and the fact that needed quantities like temperature and pressure are explicit computeable this new variable is a good choice

for the implementation of moist atmospheric models.

Beside the work presented here, the two developed classes have also been the groundwork for an implementation of a metropolis algorithm for water ice structure analysis during the master thesis of Zierenberg [2010]. During the master thesis of Schierz [2013] the classes were used for the calculation of equipotential surface and the fugacity expansion coefficients in the porous material ZIF-11. In this context the first transition of those classes to OpenCL were realized as well. Also an experimental particle system model using a Lenard-Jones potential approach was developed. Those applications show the applicability of GPGPU and the developed classes in other physical contexts as well.

Future development will lay more emphasis on multi GPU environments and the nesting of the ASAMgpu into a larger scale regional weather model, like the WRF for example. First steps into this direction were already done. Usage of multiple GPUs is possible but relativly slow because of the bandwith of the PCIe bus. This bandwith has doubled with the invention of PCIe 3.0, so with newer hardware those multi GPU setups become more interesting. In the context of the HOPE campaign in Melpitz during September 2013 an experimental model setup using boundary conditions from an external operational weather model (the WRF by Janek Zimmer, http://www.modellzentrale.de) was build up, with some first and quite promising results.

Appendices

A Listings: OpenGL context

```
1  #include <GL/glew.h>
2  #include <GL/gl.h>
3  #include <GL/glu.h>
4  #include <SDL/SDL.h>
5  #include "iostream"
6  using namespace std;
7  #include "openglcontext.h"
8
9  openglcontext::openglcontext(int sizexn, int sizeyn)
10 {
11  sizex=sizexn;sizey=sizeyn;
12  GLint intbuf;
13  SDL_Init(SDL_INIT_VIDEO|SDL_INIT_TIMER);
14  SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 8 );
15  SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 8 );
16  SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 8 );
17  SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
18  SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );
19  SDL_SetVideoMode( sizex, sizey, 32, SDL_OPENGL|SDL_RESIZABLE);
        //|SDL_FULLSCREEN
20  GLenum err=glewInit();
21  if(err!=GLEW_OK){cout << "glew error:" << glewGetErrorString(
        err) << "\n";}
22  glDisable(GL_DEPTH_TEST);
23  glDepthFunc(GL_LESS);
24  glEnable(GL_TEXTURE_2D);
25  glDisable(GL_BLEND);
26  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
27  glMatrixMode(GL_PROJECTION);
28  glLoadIdentity();
29  glViewport(0.,0.,sizex,sizey);
30  gluOrtho2D(0,1,0,1);
31  glMatrixMode(GL_MODELVIEW);
32  glLoadIdentity();
33  glClearColor(0.0,0.0,0.0,0.0);
34  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
35 }
```

Listing 19 *openglcontext* implementation (openglcontext.cpp)

B Listings: Shallow Water Example

```
1 #include <SDL/SDL.h>
2 #include <stdlib.h>
3 #include "openglcontext.h"
4 #include "shader.h"
5 #include "texture.h"
6
7 int sizex=512;
8 int sizey=256;
9 float dt=0.015;
10
11 shader *s_plot;      // shaders for plotting
12 shader *s_step;
13 shader *s_drop;
14 shader *s_rhs;
15
16 texture *t0;         // texture buffers for data
17 texture *t1;
18 texture *temp;
19 texture *rhs;
20 int frame=0;
21
22 int main()
23 {
24  openglcontext context(2.*sizex,2.*sizey);
25  t0 =new texture(sizex,sizey,1);
26  t1 =new texture(sizex,sizey,1);
27  rhs=new texture(sizex,sizey,1);
28
29  for(int x=0;x<sizex;x++)
30  for(int y=0;y<sizey;y++)
31   t0->set(x,y,0.0,0.0,5.0);
32  t0->ram2gpu();
33
```

```
34  glEnable(GL_POINT_SMOOTH); // tries to produce circular
        perturbations if driver supports it
35  glPointSize(9);             // size of droplet perturbations
36
37  s_plot =new shader(NULL,"shaders/plot");       // load shader
        for visualization
38  s_rhs  =new shader(NULL,"shaders/rhs");        // load shader
        for rhs computation
39  s_step =new shader(NULL,"shaders/step");       // load shader
        for partial runge kutta timestep
40  s_drop =new shader(NULL,"shaders/drop");       // load shader
        for droplet perturbation of height field
41
42  s_rhs->bind();
43  glUniform1iARB(glGetUniformLocationARB(s_rhs->Handle,"sizex"),
        sizex);
44  glUniform1iARB(glGetUniformLocationARB(s_rhs->Handle,"sizey"),
        sizey);
45
46  for(int i=0;i<5000;i++)
47  {
48   frame++;
49
50   glViewport(0.,0.,sizex,sizey);
51
52   if(frame%100==0 && frame<1000)       // apply droplet
         perturbation in every 5th frame
53   {
54    s_drop->tout[0]=t0;
55    s_drop->bind();
56    glBegin(GL_POINTS);
57     glColor3f(1.0,0.0,0.0); // random height
58     //glVertex2f(float(rand())/float(RAND_MAX),float(rand())/
          float(RAND_MAX)); // random position
59     glVertex2f(float(int(float(rand())/float(RAND_MAX)*sizex))/
          float(sizex),float(int(float(rand())/float(RAND_MAX)*
          sizey))/float(sizey)); // random position
60     //glVertex2f(0.5,0.5); // random position
61    glEnd();
```

```
62    }
63
64    ////////////////////////////////Start Runge-Kutta
65
66    s_rhs->t[0]=t0;
67    s_rhs->tout[0]=rhs;
68    s_rhs->bind();
69    s_rhs->step();
70
71    s_step->t[0]=t0;
72    s_step->t[1]=rhs;
73    s_step->tout[0]=t1;
74    s_step->bind();
75    glUniform1fARB(glGetUniformLocationARB(s_step->Handle,"dt"),dt
          /3.);
76    s_step->step();
77
78    s_rhs->t[0]=t1;
79    s_rhs->tout[0]=rhs;
80    s_rhs->bind();
81    s_rhs->step();
82
83    s_step->t[0]=t0;
84    s_step->t[1]=rhs;
85    s_step->tout[0]=t1;
86    s_step->bind();
87    glUniform1fARB(glGetUniformLocationARB(s_step->Handle,"dt"),dt
          /2.);
88    s_step->step();
89
90    s_rhs->t[0]=t1;
91    s_rhs->tout[0]=rhs;
92    s_rhs->bind();
93    s_rhs->step();
94
95    s_step->t[0]=t0;
96    s_step->t[1]=rhs;
97    s_step->tout[0]=t1;
98    s_step->bind();
```

```
 99    glUniform1fARB(glGetUniformLocationARB(s_step->Handle,"dt"),dt
       );
100    s_step->step();
101
102    temp=t0;t0=t1;t1=temp;
103    ////////////////////////////////End Runge Kutta scheme
104    ////////////////////////////////Start Visualization
105
106    s_plot->t[0]=t1;
107    s_plot->tout[0]=NULL;
108    s_plot->bind();
109
110    glMatrixMode(GL_PROJECTION);
111    glLoadIdentity();
112    glViewport(0.,0.,2.*sizex,2.*sizey);
113    gluOrtho2D(0,1,0,1);
114    glMatrixMode(GL_MODELVIEW);
115    glLoadIdentity();
116
117    glClearColor(0.0,0.0,0.0,0.0);
118    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
119
120    glBegin(GL_QUADS);
121     glColor4f(1.0,1.0,1.0,1.0);
122     glTexCoord2f(0.0,1.0);glVertex2f(0.0,0.0);
123     glTexCoord2f(1.0,1.0);glVertex2f(1.0,0.0);
124     glTexCoord2f(1.0,0.0);glVertex2f(1.0,1.0);
125     glTexCoord2f(0.0,0.0);glVertex2f(0.0,1.0);
126    glEnd();
127
128    SDL_GL_SwapBuffers();
129    ////////////////////////////////End Visualization
130  }
131 }
```

Listing 20 Main source code for the shallow water example

```
1 uniform sampler2D t0; // incoming data
2 uniform int sizex;    // size of incoming texture
3 uniform int sizey;    // size of incoming texture
```

```
4
5  void main()
6  {
7   // definition of variables
8   vec2 pos,posxl,posxr,posyl,posyr,posxll,posxrr,posyll,posyrr;
9   vec4 c,cxl,cxr,cyl,cyr,cxll,cxrr,cyll,cyrr,rhs;
10  vec4 Fx,Fz,Fy;
11  vec4 Fxlp,Fxlm,Fxrp,Fxrm;
12  vec4 Fylp,Fylm,Fyrp,Fyrm;
13  float Gx,Gy;
14  float Gxlp,Gxlm,Gxrp,Gxrm;
15  float Gylp,Gylm,Gyrp,Gyrm;
16  float uxl,uxr,vyl,vyr;
17
18  // unit vectors to calculate position of neighbour cells
19  vec2 dxt=vec2(1./float(sizex),0.0);
20  vec2 dyt=vec2(0.0,1./float(sizey));
21
22  // constants
23  const float dx=0.25;
24  const float dy=0.25;
25  const float g=9.81;
26  const float b=0.1;
27
28  // get stencil positions
29  pos   =gl_TexCoord[0].xy;
30  posxl =pos-dxt;
31  posxr =pos+dxt;
32  posyl =pos-dyt;
33  posyr =pos+dyt;
34  posxll=pos-2.*dxt;
35  posxrr=pos+2.*dxt;
36  posyll=pos-2.*dyt;
37  posyrr=pos+2.*dyt;
38
39  // get data for stencil
40  c   =texture2D(t0,    pos);
41  cxl =texture2D(t0,  posxl);
42  cxr =texture2D(t0,  posxr);
```

```
43  cxll=texture2D(t0, posxll);
44  cxrr=texture2D(t0, posxrr);
45  cyl =texture2D(t0,  posyl);
46  cyr =texture2D(t0,  posyr);
47  cyll=texture2D(t0, posyll);
48  cyrr=texture2D(t0, posyrr);
49
50  uxl =texture2D(t0, posxl).g;
51  uxr =texture2D(t0, posxr).g;
52  vyl =texture2D(t0, posyl).b;
53  vyr =texture2D(t0, posyr).b;
54
55  // advection
56  Fx=-(uxr*cxr-uxl*cxl)/(2.*dx);
57  Fy=-(vyr*cyr-vyl*cyl)/(2.*dy);
58
59  // gravitational forcing
60  Gx=-g*(cxr.r-cxl.r)/(2.*dx)-b*c.g;
61  Gy=-g*(cyr.r-cyl.r)/(2.*dy)-b*c.b;
62
63  // return sum of all forcings
64  gl_FragColor = Fx+Fy+vec4(0.0,Gx,Gy,0.0);
65 }
```

Listing 21 Shader source code to compute source terms (RHS)

```
1  uniform sampler2D t0;
2  uniform sampler2D t1;
3  uniform float dt;
4
5  void main()
6  {
7   vec4    c=texture2D(t0,vec2(gl_TexCoord[0]));
8   vec4 rhs=texture2D(t1,vec2(gl_TexCoord[0]));
9
10  gl_FragData[0]=c+dt*rhs;
11 }
```

Listing 22 Shader source code for one Euler timestep

```
1  uniform sampler2D t0;
2
3  void main ()
4  {
5   vec4    c=texture2D(t0,vec2(gl_TexCoord[0]));
6   gl_FragColor=c+gl_Color;
7  }
```

Listing 23 Shader source code for perturbation of heightfield

```
1  uniform sampler2D t0;
2
3  void main ()
4  {
5   vec4 c=texture2D(t0,vec2(gl_TexCoord[0]));
6
7   // map value to color (dark blue to white)
8   gl_FragData[0]=vec4(5.*(c.r-5.),5.*(c.r-5.),0.5+2.5*(c.r-5.)
        ,1.0);
9  }
```

Listing 24 Shader source code for visualisation

References

Abbe, C., editor (1910). *The mechanics of the earth's atmosphere : A collection of translations; third collection.* Smithsonian Inst., Washington, DC. Bandunterteilung auf Seite V und Ersch.jahr auf Vortitelbl. d. 1. Teils.

Ansmann, A., Petzold, A., Kandler, K., Tegen, I., Wendisch, M., Müller, D., Weinzierl, B., Müller, T., and Heintzenberg, J. (2011). Saharan mineral dust experiments samum–1 and samum–2: what have we learned? *Tellus B*, 63(4):403–429.

Bryan, G. H. and Fritsch, J. M. (2002). A benchmark simulation for moist nonhydrostatic numerical models. *Monthly weather review*, 130(12):2917–2928.

Engelmann, R., Ansmann, A., Horn, S., Seifert, P., Althausen, D., Tesche, M., Esselborn, M., Fruntke, J., Lieke, K., Freudenthaler, V., and Gross, S. (2011). Doppler lidar studies of heat island effects on vertical mixing of aerosols during samum–2. *Tellus B*, 63(4).

GEWEX Cloud System Science Team (1993). The gewex cloud system study (gcss). *Bulletin of the American Meteorological Society*, 74(3):387–399.

Heus, T., van Heerwaarden, C. C., Jonker, H. J. J., Pier Siebesma, A., Axelsen, S., van den Dries, K., Geoffroy, O., Moene, A. F., Pino, D., de Roode, S. R., and Vilà-Guerau de Arellano, J. (2010). Formulation of and numerical studies with the dutch atmospheric large-eddy simulation (dales). *Geoscientific Model Development Discussions*, 3(1):99–180.

Hickel, S. (2008). Implicit turbulence modeling for large-eddy simulation. *PhD thesis, Universität München.*

Horn, S. (2012). Asamgpu v1.0 – a moist fully compressible atmospheric model using graphics processing units (gpus). *Geoscientific Model Development*, 5(2):345–353.

Jebens, S., Knoth, O., and Weiner, R. (2009). Explicit two-step peer methods for the compressible euler equations. *Monthly Weather Review*, 137(7):2380–2392.

Jiang, H. and Cotton, W. R. (2000). Large eddy simulation of shallow cumulus convection during bomex: Sensitivity to microphysics and radiation. *Journal of the Atmospheric Sciences*, 57(4):582–594.

Manneville, P. (2006). Rayleigh-bénard convection: Thirty years of experimental, theoretical, and modeling work. In *Dynamics of Spatio-Temporal Cellular Structures*, pages 41–65. Springer.

Matheou, G., Chung, D., Nuijens, L., Stevens, B., and Teixeira, J. (2011). On the fidelity of large-eddy simulation of shallow precipitating cumulus convection. *Monthly Weather Review*, 139(9):2918–2939.

Michalakes, J. and Vachharajani, M. (2008). Gpu acceleration of numerical weather prediction. *Parallel Processing Letters*, 18(04):531–548.

Moeng, C., Dudhia, J., Klemp, J., and Sullivan, P. (2007). Examining two-way grid nesting for large eddy simulation of the pbl using the wrf model. *Monthly weather review*, 135(6):2295–2311.

Morrison, H. and Grabowski, W. W. (2008). Modeling supersaturation and subgrid-scale mixing with two-moment bulk warm microphysics. *Journal of the Atmospheric Sciences*, 65(3):792–812.

Nuijens, L. (2010). Precipitating shallow cumulus convection. *Ph.D. thesis*.

Raasch, S. and Schröter, M. (2001). Palm — a large-eddy simulation model performing on massively parallel computers. *Meteorologische Zeitschrift*, 10(5):363–372.

Rauber, R. M., Ochs III, H. T., Di Girolamo, L., Göke, S., Snodgrass, E., Stevens, B., Knight, C., Jensen, J., Lenschow, D., Rilling, R., et al. (2007). Rain in shallow cumulus over the ocean: The rico campaign. *Bulletin of the American Meteorological Society*, 88(12):1912–1928.

Richardson, L. (1922). *Weather prediction by numerical process*. University Press.

Schalkwijk, J., Griffith, E. J., Post, F. H., and Jonker, H. J. (2012). High-performance simulations of turbulent clouds on a desktop pc: Exploiting the gpu. *Bulletin of the American Meteorological Society*, 93(3):307–314.

Schierz, P. (2013). Investigation of adsorption and diffusion of hydrogen guest molecules in the metal-organic framework zif-11 by computer simulations. *Master Thesis, University of Leipzig*.

Seifert, A. and Beheng, K. (2006). A two-moment cloud microphysics parameterization for mixed-phase clouds. part 1: Model description. *Meteorology and atmospheric physics*, 92(1-2):45–66.

Siebesma, A. P., Bretherton, C. S., Brown, A., Chlond, A., Cuxart, J., Duynkerke, P. G., Jiang, H., Khairoutdinov, M., Lewellen, D., Moeng, C.-H., et al. (2003). A large eddy simulation intercomparison study of shallow cumulus convection. *Journal of the Atmospheric Sciences*, 60(10):1201–1219.

Skamarock, W. C. and Klemp, J. B. (2008). A time-split nonhydrostatic atmospheric model for weather research and forecasting applications. *Journal of Computational Physics*, 227(7):3465 – 3485. ¡ce:title¿Predicting weather, climate and extreme events¡/ce:title¿.

Slawinska, J., Grabowski, W. W., Pawlowska, H., and Morrison, H. (2011). Droplet activation and mixing in large-eddy simulation of a shallow cumulus field. *Journal of the Atmospheric Sciences*, 69(2):444–462.

Steppeler, J., Doms, G., Schättler, U., Bitzer, H., Gassmann, A., Damrath, U., and Gregoric, G. (2003). Meso-gamma scale forecasts using the nonhydrostatic model lm. *Meteorology and Atmospheric Physics*, 82(1-4):75–96.

Stevens, B., Lenschow, D. H., Vali, G., Gerber, H., Bandy, A., Blomquist, B., Brenguier, J., Bretherton, C., Burnet, F., Campos, T., et al. (2003).

Dynamics and chemistry of marine stratocumulus-dycoms-ii. *Bulletin of the American Meteorological Society*, 84(5):579–593.

Stevens, B., Moeng, C.-H., Ackerman, A. S., Bretherton, C. S., Chlond, A., de Roode, S., Edwards, J., Golaz, J.-C., Jiang, H., Khairoutdinov, M., et al. (2005). Evaluation of large-eddy simulations via observations of nocturnal marine stratocumulus. *Monthly weather review*, 133(6):1443–1462.

Straka, J., Wilhelmson, R. B., Wicker, L. J., Anderson, J. R., and Droegemeier, K. K. (1993). Numerical solutions of a non-linear density current: A benchmark solution and comparisons. *International Journal for Numerical Methods in Fluids*, 17(1):1–22.

VanZanten, M. C., Stevens, B., Nuijens, L., Siebesma, A. P., Ackerman, A., Burnet, F., Cheng, A., Couvreux, F., Jiang, H., Khairoutdinov, M., et al. (2011). Controls on precipitation and cloudiness in simulations of trade-wind cumulus as observed during rico. *Journal of Advances in Modeling Earth Systems*, 3(2).

Wicker, L. J. and Skamarock, W. C. (1998). A time-splitting scheme for the elastic equations incorporating second-order runge-kutta time differencing. *Monthly weather review*, 126(7):1992–1999.

Wicker, L. J. and Skamarock, W. C. (2002). Time-splitting methods for elastic models using forward time schemes. *Monthly Weather Review*, 130(8):2088–2097.

Zierenberg, J. (2010). Tip4p water model in the ice ih configuration. *Master Thesis, University of Leipzig*.

Danksagung

An dieser Stelle möchte ich zuerst Herrn Dr. Oswald Knoth meinen Dank aussprechen. Er begleitet meine wissenschaftliche Laufbahn nun schon seit meiner Diplomarbeit und stand mir immer gern mit Rat und Tat zu Seite.

Des Weiteren möchte ich Herrn Prof. Dr. Heinzenberg und Herrn Prof. Dr. Macke danken, die, als Leiter des Instituts für Troposphärenforschung, stets offen für Ideen waren und die Entwicklung des Modells unterstützt haben. Gleicher Dank gebührt Herrn, Prof. Dr. Renner und Frau Prof. Dr. Tegen, die, als Abteilungsleiter der Abteilung für Modellierung, für eine positive Arbeitsatmosphäre und die nötige finanzielle Basis bis zum Abschluss dieser Arbeit gesorgt haben. Anteil an diesem angenehmen Arbeitsumfeld haben natürlich auch alle Kollegen und Freunde die am Tropos tätig sind oder waren.

Weiterhin gilt mein Dank allen Freunden aus meinem privaten Umfeld, sowie meiner Familie.

Wissenschaftlicher Werdegang

2000-2006	Physikstudium an der Universität Leipzig, Diplom Physik
2003-2006	Studentische Hilfskraft in verschiedenen Abteilungen der Universität Leipzig
2006-2008	Wissenschaftlicher Mitarbeiter am Institut für Meteorologie an der Universität Leipzig (Projekt: "Biowind", Ausbreitungsmodellierung von Pflanzendiasporen in der Grenzschicht)
2008-2010	Wissenschaftlicher Mitarbeiter/Promotionsstudent am Institut für Troposphärenforschung Leipzig (Projekte: EUCAARI, PAKT)
2010-2012	Wissenschaftlicher Mitarbeiter/Promotionsstudent am Institut für Troposphärenforschung Leipzig (Abteilungsübergreifendes Projekt: "Lidar begleitende Large Eddy Simulationen mit einem schnellen, hochauflösenden GPU-Modell")
2012-2014	Wissenschaftlicher Mitarbeiter/Promotionsstudent am Institut für Troposphärenforschung Leipzig (Projekt HDCP2/HOPE: "Kampagnen begleitende Large Eddy Simulationen mit einem in einem regionalen Wettermodell genesteten GPU-Modell", "Auf dem Weg zu einem operationellen LES-Modell")

Veröffentlichungen und Tagungsbeiträge

Horn, S.: "ASAMgpu V1.0 - a moist fully compressible atmospheric model using graphics processing units (GPUs)", Geosci. Model Dev. Discuss., 4, 2635-2660, doi:10.5194/gmdd-4-2635-2011, 2011.

Engelmann, R., Ansmann, A., Horn, S., Seifert, P., Althausen, D., Tesche, M., Esselborn, M., Fruntke, J., Lieke, K., Freudenthaler, V., Gross, S.: "Doppler lidar studies of heat island effects on vertical mixing of aerosols during SAMUM-2", Tellus B, North America, 63, doi: 10.1111/j.1600-0889.2011.00552.x, 2011.

Horn, S., Raabe, A., Wil, H., Tackenberg, O.: "TurbSeed - A model for wind dispersal of seeds in turbulent currents based on publicly available climate data", Ecological Modelling, Volumes 237-238, Pages 1-10, ISSN 0304-3800, doi: 10.1016/j.ecolmodel.2012.04.009, 2012.

Horn, S., Wilsdorf, M., Raabe, A.: "Räumlich explizite Modellierung der Ausbreitung von Pflanzen Diasporen", Wissenschaftliche Mitteilungen des LIM, Volume 41, Pages 85-98, 2007.

Vorträge

Biowind Workshop, Langen, 2007,
A new approach for modelling seed dispersal by wind

Doktoranden Seminarvorträge, IfT, Leipzig, 2007-2010,
Large Eddy Simulations (LES) of stratocumulus cloud fields

Seminarvortrag, Universität Halle, 2009,
Feuchte Grenzschicht Simulation mit OpenGL Shadern

Seminarvortrag, MPI Hamburg, 2010,

Large Eddy Simulation on GPU's using OpenGL/GLSL

Facing the Multicore-Challenge, Heidelberg, 2010,
Moist Planetary Boundary Layer Simulation Using OpenGL and GLSL

ModelCare, UFZ Leipzig, 2011,
Modeling and Visualisation of cloud systems in the planetary boundary layer using GPUs

ICCP, Leipzig, 2012,
Large Eddy Simulations of aerosol and resolution dependence of open cell structures in stratocumulus cloud layers using GPU

Poster

EGU, Wien, 2007,
Frequency domain analysis and modeling of velocity in the surface layer to develop a trajectory diaspore dispersal model

DACH, Hamburg, 2007,
Modellierung der Windausbreitung von Diasporen in strukturiertem Gelände

EUCAARI IMPACT-LONGREX joint workshop, Toulouse, 2008,
LES modelling of warm clouds using a GPU

Proceedings of EUCAARI annual meeting, Stockholm, 2009,
LES modelling of warm clouds using a GPU

METTOOLS VIII, Leipzig, 2012,
Modellierung komplexer meteorologischer Strömungen mittels Grafikkarten - das Modell ASAMgpu