Design and Performance Evaluation of Snow Cover Computing on GPUs

LADISLAV HURAJ¹, VLADIMÍR SILÁDI², JOZEF SILÁČI²
¹Department of Applied Informatics
University of SS. Cyril and Methodius in Trnava
Nám. J. Herdu 2, 917 01 Trnava
SLOVAK REPUBLIC
ladislav.huraj@ucm.sk
²Department of Computer Science
University of Matej Bel
Tajovskeho 40
SLOVAK REPUBLIC
{siladi, silaci}@fpv.umb.sk

Abstract: The global warming has an effect on changes of a snow cover over wintertime. This effect is observed in Slovak ski resorts, too. A prediction of these trends is important to build new and keep the existing ski resorts. We are able to analyze the snow cover depth in detail. This analysis is based on many continuous observations and measurements at the specific climatologic stations of Slovak Hydrometeorogical Institute. However, these climatologic stations do not render accurately all ski places, which could be examined. The aim of this work is the depth of the snow cover computing in the desired point based on the geographical characteristics of a specific geographical point in a modeled area. The main characteristics of the computing is the fact that it is time-consuming. One solution is a utilization of graphics processing units (GPUs) where the availability of enormous computational performance of easily programmable GPUs can rapidly decrease time of computing.

In our article we demonstrate how to deploy the CUDA architecture, which utilizes the powerful parallel computation capacity of GPU, to accelerate computational process of snow cover depth using the inverse-distance weighting (IDW) method. The outputs are visualized by the Grass GIS tool.

Key-Words: GPGPU, Cuda, snow cover depth

1 Introduction
A big attention in the world is given to the research of prognosis if the depth of snow cover depends on influence of global warming. The high performance computing resources (supercomputers, computer grids ....) are used for this purpose [1,2,6]. In most cases, the processing relates to a much larger area than continents or countries [2,3,4]. Determination of the snow cover depth in a defined territory seems to be problematic because of absence of rain gauge stations. Therefore, the computationally time-consuming interpolation methods for determination of the snow cover depth are necessary where the data from nearby rain gauge stations are used.

Our objective in this paper is to develop a GPGPU (General-Purpose computing on Graphics Processing Units) design that can be used for implementation of snow cover modeling and for acceleration of the computing. The utilization of GPGPU will be tested for various methods, inverse-distance weighting method (IDW) being the first one.

Graphics Processing Units (GPUs) are widely used among researchers and developers as accelerators for applications outside the domain of traditional computer graphics. This trend largely results from the great improvements in GPU programmability [7]. CUDA (Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA which presents to the programmer a fairly generic abstraction of a manycore architecture supporting fine-grained parallelism. CUDA and the GPU therefore provide massive, general purpose parallel computation resources with the potential for speedups of data processing.

In our article the processed data by GPGPU is climatologic data. For the Slovak Republic are available widely relevant data monitored by Slovak Hydrometeorological Institute, which has been recording the trends of snow conditions on the specific hydrometeorological stations for at least 20 years. Using these data, we have tried to generate computer models of the depth of snow cover for a defined area during the period of time and to visualize the data by geographic information systems (GIS).

The rest of the paper is organized as follows. First we review the CUDA programming model briefly in...
section 2. Section 3 introduces the background of snow cover modeling. Section 4 describes the algorithms and their mapping on the GPU with experimental results. The conclusion comes in section 5 with an outlook to further work.

2 CUDA

The Compute Unified Device Architecture (CUDA) allows developers to use the C programming language for the development of general-purpose applications using fine-grain parallelism. CUDA is currently supported only on NVIDIA GPUs, but recent work has shown the viability of compiling CUDA programs for performance on multi-core processors [8]. A simple extension to C had invited that more non-graphics developers port their existing applications to CUDA. CUDA consists of a runtime library and an expanded version of C. CUDA gives developers access to the native instruction set and memory of the parallel computational elements in CUDA GPUs. It includes the CUDA Instruction Set Architecture (ISA) and the parallel compute engine in the GPU [9].

A single source program contains both the host (CPU) code and the device (GPU) code which are automatically separated and compiled by the CUDA compiler tool chain, Figure 1.

![Fig. 1 Heterogeneous Programming in CUDA](image)

CUDA is based on the notion of a kernel function, which is a single routine that is invoked concurrently across many thread instances; a software controlled scratchpad, which CUDA calls the “shared memory”, in a Single Instruction Multiple Data (SIMD) fashion for each SIMD core; and barrier synchronization. Each GPU thread is a fully independent, scalar, and can execute arbitrary code and access arbitrary addresses. Moreover, each of the GPU threads is given a unique ID that is accessible within the kernel through the built-in threadIdx variable [10,11].

CUDA presents a virtual machine consisting of an arbitrary number of streaming multiprocessors (SMs), which appear to be 32-wide SIMD cores with a total of up to 512 thread contexts. Kernels are applied to a 2D grid that is divided into as many as 64K 3D thread blocks. Each thread block is mapped in its entirety and executes to completion on an arbitrary SM. All thread blocks in a kernel run to completion before a subsequent kernel may start, offering a global memory fence [10,11].

3 Snow Cover Modeling

The depth of the snow cover is a very variable meteorological element in the landscape. It depends on many factors, mainly on snow precipitation, altitude, air temperature, profile of the relief, solar power, cloudiness, air temperature inversion, etc. The measurement of the snow cover is taken by meteorological, climatologic and precipitation stations. The total depth of snow is measured and stored, i.e. the dept of snow and the depth of new snow cover. We are able to analyze the depth of the snow cover in detail.

The analysis is based on many continuous observations and measurements at the specific climatologic stations. Geographically we can strictly characterize all these gauging places by the altitude, latitude and longitude, as well as by the detailed characteristics of the relief shape. The aim of this work is the depth of the snow cover computing in the arbitrary point based on the geographical characteristics of a specific geographical point in a modeled area. The result is derived from the available data, which have been obtained from metrological stations, climatologic stations and rain gauge stations from a defined landscape area.

![Fig. 2 Location of the 17 meteorological stations](image)
As an example of the application of our designed method the geomorphological entity Zvolenská kotlina as a part of Slovak Republic has been chosen, which is exactly defined by its borders. We use the digital terrain model of this entity, which has been done. The 17 meteorological stations of Slovak Hydrometeorological Institute are situated in this defined area and their long-term measurements are available for our research, Figure 2. We decided for the period of years 1990–2009. The input data are stored in the large matrices. The output values depend on the time-consuming computing process and CUDA was deployed to accelerate these operations.

3.1 Inverse-distance Weighting Method
We used an inverse-distance-weighting algorithm to interpolate the snow cover measurements. This deterministic model is relatively fast and easy to compute, and straightforward to interpret. The IDW method as a deterministic spatial interpolation model is one of the more popular methods adopted by geoscientists and geographers partly because it has been implemented in many GIS tools [12]. The general assumption of this method is that the attribute values of any given pair of points are related to each other, but their similarity is inversely related to the distance between the two locations. The general idea of IDW is that the attribute value of an un-sampled point is the weighted average of known values within the neighborhood, and the weights are inversely related to the distances between the location of un-sampled point and the location of its neighbors. The value of inverse-distance weight is modified by a constant power with increasing distance. This dependence can be expressed by the following relationship:

\[
y_0 = \frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} \frac{1}{d_i^p}}
\]

where \( y_0 \) is value of un-sampled point; \( d_i \) denotes the distance between un-sampled point and sampled location \( i \) and \( y_i \) is given value at sampled locations \( i \).

Moreover, the S-JTSK coordinates system was implemented and the conversion of coordinates from WGS-84 into S-JTSK was necessarily performed.

4 Design and Performance Evaluation
To write a CUDA program, we obeyed the following steps from [7] starting from a sequential version and proceeds through:

1) Identify the kernel and package it as a separate function.
2) Specify the GPU threads’ grid and partition the computation among these threads.
3) Manage data transfer between the memory of host and GPU memories before and after the kernel invocation.
4) Make memory optimizations in the kernel.
5) Make further optimizations in the kernel for optimizing the single thread performance and the level of parallelism.

In the experimental evaluation of our computing we focused on the performance improvements from CUDA implementation. We investigated two likely scenarios of use: the first, a dual-core CPU where the computations run as serial code, compared to the second one where the GPUs were implemented. The GPU, with its fitting SIMD architecture, executes these instructions much faster than the CPU.

Runtimes for all implementations were measured on a machine running Ubuntu version 9.04 with a 2.8 GHz dual-core Intel Core 2 E7400 processor and an NVIDIA GeForce 9800 GT, with NVIDIA driver version 195.36.15. The C code was compiled using GCC version 4.3.3 and the CUDA code was compiled using NVCC version 0.2.1221, CUDA Toolkit version 3.0.

The final CUDA implementation of the snow cover computing provides a speedup of 6.7x over the single-threaded C implementation and the new computation time represents only 15% of previous time necessary for computing of the interpolations.

![Fig. 3 Graphical output of snow cover computing in centimeters](image-url)
The experiments evaluate the proposed parallelization, and show faster computational times on the proposed architecture than on only CPU.

Figure 3 shows an example of graphical output of our experiment visualized by the GRASS (Geographic Resources Analysis Support System) geographical information system tool version 6.4.0. The snow cover was calculated from data from January, the 5th 2000 and the depth is visualized in centimeters.

5 Conclusion

In our article we have shown that snow cover computing benefit from using CUDA GPUs. For computing of non-cover points we have used an inverse-distance-weighting algorithm and we process meteorological data of twenty years period. The CUDA implementation provides speedups of 6.7x on the C implementation.

Overall, our experiences with CUDA show the power of the GPU as a parallel platform, and help demonstrate how to utilize the GPGPU programming for geographic data processing.

In the future, we are going to implement more efficient algorithms via implementing other interpolation methods where we suggest more rapidly speed-up of the GPGPU programming. Moreover, we also plan to transform the computation from GPGPU programming to Grid Computing.

Acknowledgements: The authors gratefully acknowledge the data support of Slovak Hydrometeorological Institute.

References:


