A High Performance GPU Implementation of Surface Energy Balance System (SEBS) based on CUDA-C

Mohammad Abouali[a,b], Joris Timmermans[c], Jose E. Castillo[a], Bob Z. Su[c]

[a] *Computational Science Research Center at San Diego State University, 5500 Campanile Drive, San Diego, CA 92182-1245, USA*
[b] *Corresponding Author, email: mabouali@sciences.sdsu.edu, Phone: (+1) (619) 594-3430, Fax: (+1) (619) 594-2459*
[c] *Faculty of Geo-information Science and Earth Observation at University of Twente, Hengelosestraat 99, 7514 AE, Enschede, The Netherlands.*

Abstract

This paper introduces a new implementation of the Surface Energy Balance System (SEBS) algorithm harnessing the many cores available on Graphics Processing Units (GPUs). This new implementation uses Compute Unified Device Architecture C (CUDA-C) programming model and is designed to be executed on a system equipped with NVIDIA®'s graphic cards. The output of the new implementation is compared to a MATLAB code that has already been fully tested in the Water Cycle Multimission Observation Strategy (WACMOS) project. The code is timed against both MATLAB and a purely high-performance C implementation of the same algorithm. The code has been tested on several different NVIDIA® cards, with different compute capabilities. The authors have decided to provide the entire source code to the scientific community free of charge; hence, at the end, the instruction on how to obtain the code is also presented.

Keywords: Evapotranspiration, Remote Sensing, Surface Energy Balance System, SEBS, CUDA-C, High Performance Computing, NVIDIA®, GPU

1. Introduction

EvapoTranspiration (ET) is a portmanteau word, describing the combined amount of water evaporated and transpired by canopies and land surfaces. ET represents a direct feedback of moisture to the atmosphere and

is of utmost importance in the understanding of the terrestrial climate system[1,2,3] and therefore it is of interest in several applications, such as Drought Assessment, Agricultural Irrigation Management, Weather Forecasting, Hydrological Modeling, and Climate Simulations and Predictions.

Several types of ET, found in the literatures, are (1) potential (2) reference or (3) actual. Potential ET (PET) provides an estimate of the total capacity or energy available to evapotranspire water, assuming there is no limitation in water resources. Reference ET refers to the amount of water evaporated by a reference crop (usually short grass) under the prevailing meteorological conditions. The goal of estimating Actual ET (AET) is to come up with an estimate of the real or actual state of the system.

There are several methods to calculate AET[4,5,6], which have been already reviewed in[7] and[8]. Not only do these algorithms tackle a large set of coupled nonlinear equations, they also require a large set of input variables; hence, too much computational time is needed. This paper focuses on SEBS and its implementation on GPU using CUDA-C in order to increase its performance by decreasing the total computation time needed.

2. Surface Energy Balance System (SEBS)

SEBS uses the surface energy balance (SEB) equation to calculate ET. However, contrary to other algorithms, that calculate ET as the residual of this equation, SEBS make use of the Evaporative Fraction (EF) to estimate the actual evapotranspiration in order to account for water limiting cases. The SEB equation[7,8] describes the net radiation as the sum of four components, i.e.:

$$R_n = \lambda E + G_0 + H + S, \tag{1}$$

where R_n is the total net radiation, λE is the latent heat, G_0 is the ground heat flux, H is the sensible heat, and S is the storage of heat, $[Wm^{-2}]$. In SEBS, latent heat is calculated using:

$$\lambda E = \Lambda(R_n - G_0), \tag{2}$$

where Λ is the evaporative fraction (dimensionless) that is calculated as follows:

$$\Lambda = \Lambda_r \frac{\lambda E_{wet}}{R_n - G_0}, \tag{3}$$

and

$$\Lambda_r = 1 - \frac{H - H_{wet}}{H_{dry} - H_{wet}}. \quad (4)$$

Λ_r is called the relative evaporation (dimensionless). H_{dry} and H_{wet} are the sensible heat fluxes under the hypothetical dry and wet conditions. λE_{wet} is the evapotranspiration under the wet conditions, and in fully dry conditions, λE_{dry} is zero. For more detailed information on the SEBS algorithm, governing equations, and how to process data one can refer to journal papers and reports already published, such as[4,9,10,11].

3. SEBS Data Requirements

SEBS requires three sets of input data:

- The first set consists of remotely sensed vegetation parameters, such as: surface albedo, emissivity, and surface temperature.
- The second set includes meteorological data, such as: air temperature, wind speed, and vapor pressure.
- The third set of data includes remotely sensed parameters such as incoming longwave and shortwave radiation.

In this paper, Climate Forecast System Reanalysis (CFSR) (`http://dss.ucar.edu/pub/cfsr.html`, $\sim 0.3^\circ$ resolution), North America Land Data Assimilation System (NLDAS) (`http://ldas.gsfc.nasa.gov/nldas/`, 0.125° resolution), along with Moderate-Resolution Imaging Spectroradiometer (MODIS) products were used.

4. Graphical Processing Units and CUDA-C

High resolution estimates of ET by SEBS can be a very time consuming task. As an example, calculation of ET using a single processor over the United State of America (USA) with 1km Spatial resolution and daily temporal resolution for one year can take about 10 days (based on 25 MODIS tiles). This will increase exponentially with decreasing spatial resolution. Therefore, it is important to provide a high-performance code that can reduce the computation time considerably.

The SEBS algorithm requires many inputs; however, once all the required input variables are provided in a single grid cell (pixel), there would be no inter-pixel communication between one pixel and its surrounding pixels. Thus, a domain-decomposition approach to parallelize the SEBS algorithm would be the best and the easiest approach. In a domain-decomposition approach using central processing units (CPUs) the entire computational domain is divided into smaller parts, and each part is assigned to a single core. If one needs higher speedup, more processor nodes must be provided. This means that multiple machines with very strong CPUs have to be networked together. However, this requires to transport data over network links and deal with the latency and overhead associated with it, not to mention the cost of descent networking devices that are required to connect these computing nodes together.

An interesting solution is to move the computation to the graphics processing units (GPUs) that exists almost on every graphic card sold these days. A GeForce GTX 580 with 3GB of DDR5 memory will provide 512 computing cores. A Tesla C2070 with 6GB of memory provides the user with 448 computing cores. All these computing cores are on a single device and there is no need to purchase any extra networking devices. Consequently the cost per computing core is way lower using GPUs rather than CPUs. Many top supercomputers are now equipped with GPUs. In fact "NVIDIA chips are now in three of the five fastest supercomputers in the world" [12]. About three years ago the first supercomputer, equipped with GPU, showed up in top 500 supercomputers list. Since then these numbers are growing[12,13]. Since then GPUs are being used in many fields including environmental modeling[14,15,16,17].

While the number of cores can be increased easily, the performance boost depends greatly on adaptability of the code to these cores. To run algorithms on the GPU cores, first the code needs to be rewritten with the programming model suitable for GPUs. There are few programming models available, that can be used to communicate with this device and perform scientific computing, known as General Purpose Graphics Processing Units (GPGPUs). OpenCL and CUDA-C are the most common ones. OpenCL targets different hardware, while CUDA is developed by NVIDIA and can be used only on machines equipped with NVIDIA's GPUs. In CUDA, one launches a computation grid, with different computing blocks in each grid, where each block holds a group of computing threads. It is possible to assign each thread a certain task. These computational grids are scheduled on GPUs to be ex-

ecuted. In this case, the grid was chosen in such a way that there is one thread for each pixel of a MODIS tile.

For more information on CUDA and its programming model one can refer to the documents provided by NVIDIA on CUDA zone (`http://developer.nvidia.com/nvidia-gpu-computing-documentation`).

5. Test Data

A test data set is prepared in order to compare the performance of different implementations of SEBS algorithm. The data set is chosen over a small region covering Prosser, WA, with 120×120 pixels, figure (1). The resolution of each pixel is $1km$. The MODIS products were the source of remote sensing data sets. The Majority of the meteorological data were collected from NLDAS, with a few exception that were collected from CFSR, including mean sea level pressure. The NCAR Command Language (NCL - `http://www.ncl.ucar.edu/`) was used to preprocess the data set and prepare the input data set for SEBS in NetCDF format. All data belongs to the 194^{th} day of year 2008. It has to be noted that this data set is 100 times smaller than a single MODIS tile and there are 25 MODIS tiles that cover the USA completely.

6. MATLAB to C

First the MATLAB codes were converted to C language.The resulting C code was optimized by changing the order of some of the calculations and reducing the number of memory access. Later, the code was compiled. Both -O2 and -O3 optimization flags were tested. The level 3 optimization flag was only slightly improving the performance. In both MATLAB and C code the time needed to read the input data from NetCDF files and storing the output was ignored and only the time needed to run SEBS algorithm was considered in the speedup calculation. Both MATLAB (Version 2011a) and C code were executed on a Macbook Pro with 8GB of DDR3 RAM and equipped with 2.66 GHz Intel Core 2 Duo CPU. The code was run several times, the lowest and the highest time were ignored and then averaged. As it is shown in figure (2), the C code already achieved a speedup of about 4.4. Since the MATLAB version was already validated as part of the WACMOS project[18,19,20,21], the output of the C code was compared against those of MATLAB. This comparison is shown in figure (3) for daily evapotranspiration.

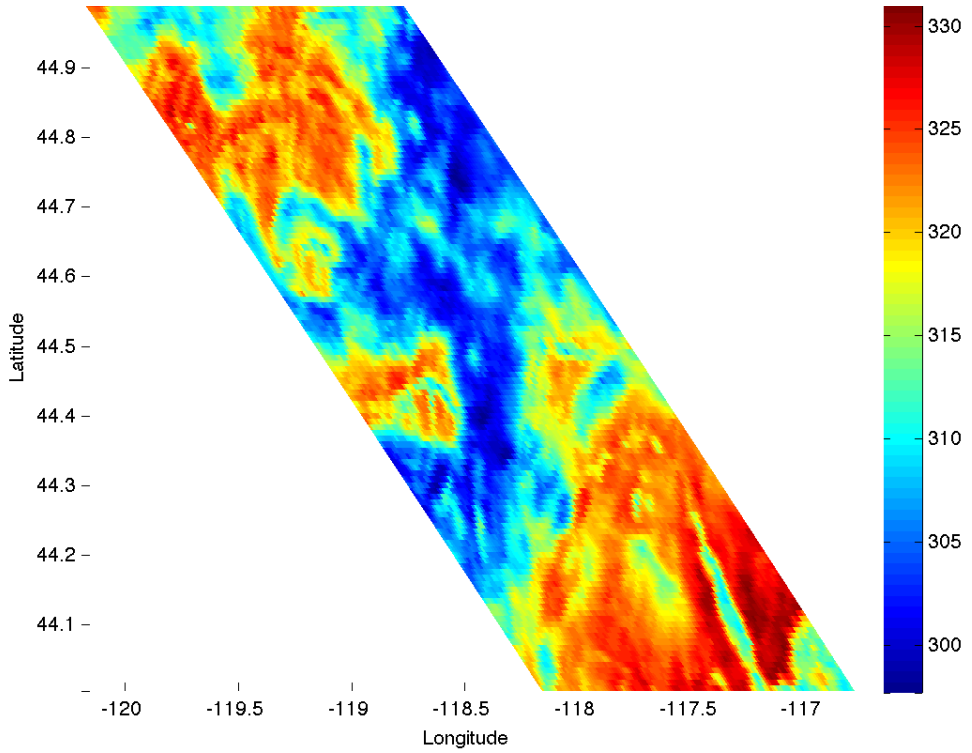

Figure 1: Land Surface Temperature [K] - MOD11A1 - Tile: h09v04 - Year 2008 - DoY 194.

7. C to CUDA-C

The next stage was implementing the SEBS algorithm using CUDA-C to be executed on GPUs. The C functions developed previously were wrapped with CUDA instructions and introduced as device functions. Later, certain global functions (CUDA kernels) were developed, which assigns one computing thread to each pixel of the input data sets. The code timed on different devices. The list of devices, the number of computing cores available, and their compute capabilities is listed in table (1).

One of the bottle-necks in GPU computing is the transfer of data from computer RAM to video memory on graphic card. Unlike C and MATLAB code that included only the computation time, the timing on CUDA-C also included the amount of time needed to upload data from RAM into the graphic card and download the results from the graphic card on to the RAM. As figure (2) shows, even on a very old graphic card, (GeForce 9400M), a

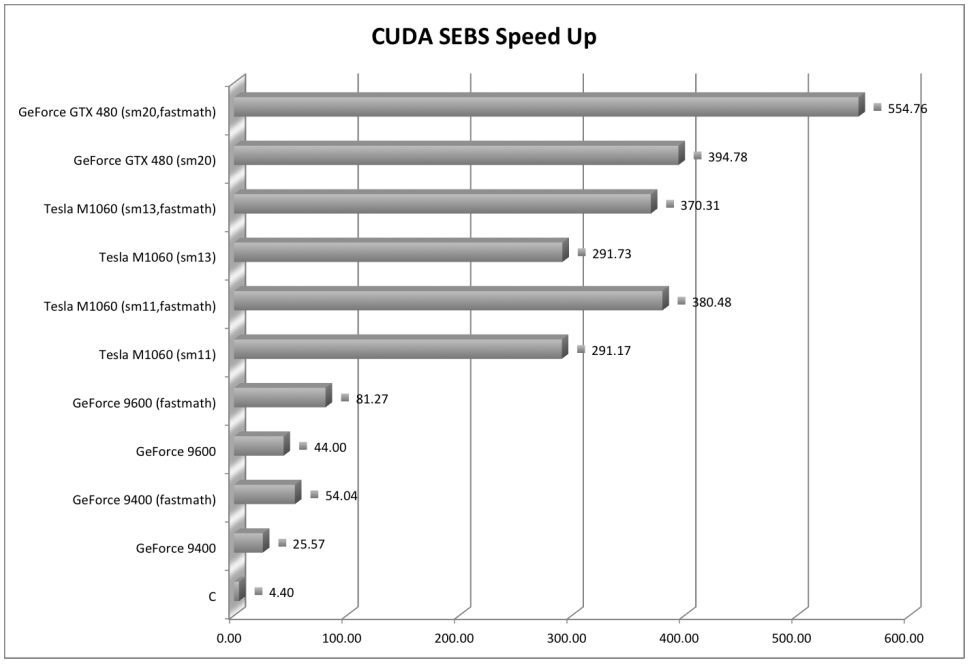

Figure 2: Speedup relative to MATLAB code.

speedup of about 25.6 relative to MATLAB code was achieved, (or 5.8 relative to C code). CUDA-C implementation was also tested on Tesla M1060 card and achieved a speedup of about 291.2 relative to MATLAB code, (or 66.2 relative to C code).

The *nvcc* compiler allows the use of '-use_fast_math' option, which forces the code to use a less accurate but much faster version of some functions, such as exp(). As figure (2) shows this will increase the speedup on GeForce 9400M from 25.6 to 54.0 (a factor of 2.1) and on Tesla card from 291.2 to 380.5 (a factor of 1.3). No sensible changes in the output accuracy were detected while using '-use_fast_math' compiler options. The output of CUDA-C version using '-use_fast_math' compared to MATLAB version is shown in figure (4). Hence, it is suggested to keep this switch, while compiling. Table (2) summarizes the performance achieved on different platforms using different compiler flags.

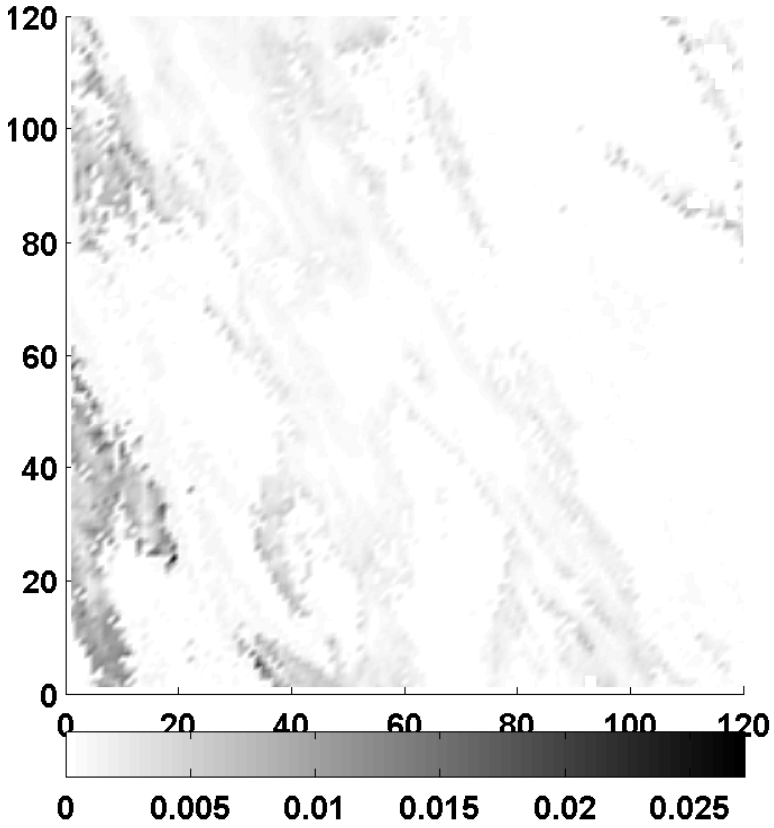

Figure 3: Difference in daily evapotranspiration $[mm \cdot day^{-1}]$ between MATLAB and C Code.

Table 1: GPU devices used in timing

Model	Memory	N. Cores	Compute Cap.
GeForce 9400M	256MB	16	1.1
GeForce 9600M	256MB	32	1.1
Tesla M1060	4GB	240	1.3
GeForce GTX 480	1.5GB	480	2.0

8. Conclusions and Suggestions

Two new implementations of the SEBS algorithm using C and CUDA-C were introduced. These implementations were tested using a small data set

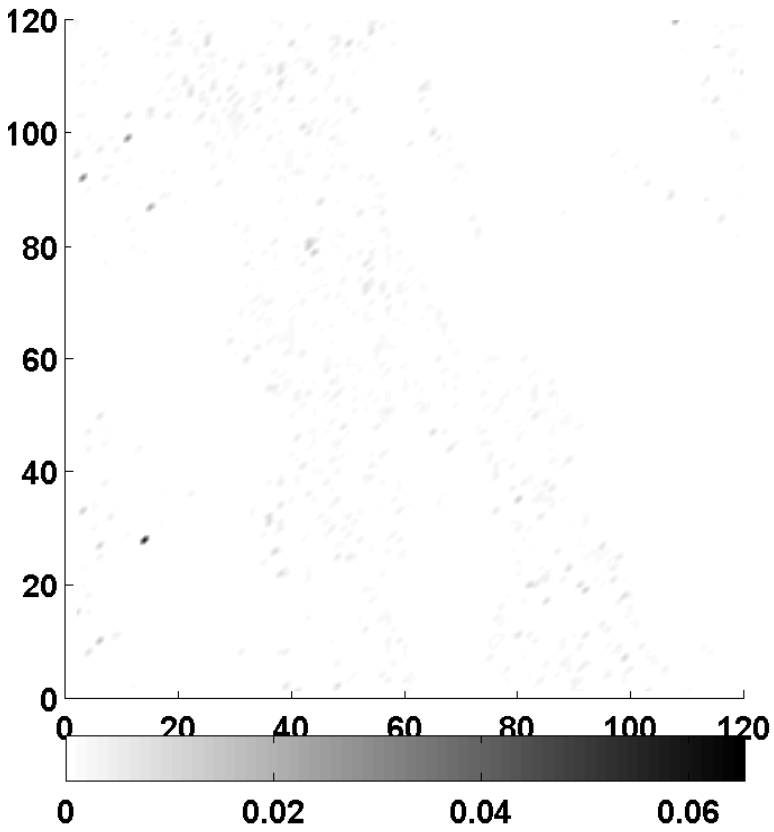

Figure 4: Difference in daily evapotranspiration $[mm \cdot day^{-1}]$ between MATLAB and CUDA-C Code.

and the codes were timed. At best case, it was possible to achieve about 554 (126.1) times speedup (going from 10 days of computation to 30 minutes). Of course the time needed to read the input data from hard drive and store it back must be added; however, these overheads must be added regardless of the implementation being used. This speedup was achieved by using only one CUDA-enabled card. To further speedup the program one can use multiple NVIDIA card. One can also make use of streams available in CUDA-C to separate the memory transfer and the computation. Here, the computation space was decomposed only spatially, but it can be also done temporally. As an example, the Computational Science Research Center (CSRC) at San Diego State University (SDSU) hosts a cluster with 12 nodes connected by InfiniBand. There are two Tesla M1060 card

Table 2: Speedup

Device	Compile Switch	Speedup Rel. to MATLAB	Speedup Rel. to C
GeForce GTX 480	f4, f3	554.76	126.1
GeForce GTX 480	f4	394.78	89.7
Tesla M1060	f1, f3	380.5	86.4
Tesla M1060	f1	291.2	66.1
Tesla M1060	f2	291.7	66.2
Tesla M1060	f2, f3	370.3	84.1
GeForce 9600M	f1, f3	84.7	19.2
GeForce 9600M	f1	43.8	9.9
GeForce 9400M	f1, f3	54.0	12.3
GeForce 9400M	f1	25.6	5.8

f1: -arch=sm_11, f2:-arch=sm_13, f3: -use_fast_math, f4:-arch=sm_20

available on each node. It is possible to combine CUDA-C with MPI and assign one node for each month and one Tesla card only for 15 days of calculation. Furthermore, it is believed that using Parallel NetCDF (`http://trac.mcs.anl.gov/projects/parallel-netcdf`) and having concurrent access to the input data sets could enhance the performance even more. The authors hope that they can address these issues in the near future.

Software Availability

- Software Name: SEBS-GPU.
- Developer: The same as the authors of this paper.
- Hardware Requirements: PC equiped with CUDA-Enabled NVIDIA Graphic card.
- Software Requirements: GCC, NVCC, NetCDF, MATLAB (only for MATLAB interface).

- Cost: free only for noncommercial use and solely for academic and research purposes.
- Programming Language: C, CUDA-C, MATLAB (only for MATLAB interface).
- Download instruction: Requires subversion. To download anonymously (no username or password required) issue the following command:

```
svn co http://sebs-gpu.googlecode.com/svn/trunk/ sebs-gpu-read-only
```

Acknowledgments

The authors would like to thank ITC (http://www.itc.nl), Enschede, the Netherlands, and CSRC (http://www.csrc.sdsu.edu), San Diego, California, USA, for funding this project and providing us with access to their computing clusters equipped with NVIDIA's Tesla cards. Furthermore, we would like to extend our thanks to Dennis Shea and Bob Dattore from the National Center for Atmospheric Research (NCAR), Boulder, Colorado, USA, for sharing their knowledge about NLDAS data set and providing us with access to CFSR data set.

References

1. Penman H. Natural evaporation from open water, bare soil and grass. *Proc Roy Soc A* 1948;193(120-146).
2. Su Z, Menenti M. Mesoscale climate hydrology: the contribution of the new onserving systems. Report USP-2,99-05; Publications of the National Remote Sensing Board (BCRS); 1999.
3. Su Z, Jacobs C. Advanced earth observation - land surface climate. USP-2, 01-02; Publications of the National Remote Sensing Board (BCRS); 2001.
4. Su Z. The surface energy balance system (sebs) for estimatio of turbulent heat fluxes. *Hydrology and Earth System Sciences* 2002;6(1):85–99.

5. Bastiaanssen W. Regionalization of surface flux densities and moisture indicators in composite terrain - a remote sensing approach under clear skies in mediterranean climates. Ph.d. thesis; Wageningen Agricultural University; The Netherlands; 1998.

6. Kustas W, Norman J. Evaluation of soil and vegetation heat flux predictions using simple two-source model with radiometric temperatures for partial canopy cover. *Agr Forest Meteorol* 1999;94:13–29.

7. Glenn E, Huete A, Hirschboeck PNK, Brown P. Integrating remote sensing and ground methods to estimate evapotranspiration. *Critical Reviews in Plant Sciences* 2007;26(139-168).

8. Kalma J, McVicar T, McCABE M. Estimating land surface evaporation: A review of methods using remotely sensed surface temperature data. *Surveys in Geophysics* 2008;29(421-469).

9. Timmermans J, Su Z. Support to science element water cycle multimission observation strategy - design definition. Tech. Rep. WACMOS-DD DDF-V1; ITC - University of Twente; The Netherlands; 2011.

10. Jia L, Su Z, van den Hurk B, Menenti M, Moene A, Bruin HD, Yrisarry J, Ibanez M, Cuesta A. Estimation of sensible heat flux using the surface energy balance system (sebs) and atsr measurements. *Phys Chem Earth* 2003;28(75-88).

11. Su H, McCABE M, Wood E. Modeling evapotranspiration during smacex: Comparing two approaches for local- and regional-scale prediction. *Journal of Hydrometeorology* 2005;6(910-922).

12. Crothers B. Why nvidia's chips can power supercomputers. 2011. URL `http://news.cnet.com/8301-13924_3-57332553-64/why-nvidias-chips-can-power-supercomputers/`.

13. Abouali M. Why moving towards gpus? 2011. URL `http://mabouali.wordpress.com/2011/12/06/why-moving-towards-gpus/`.

14. Sousa F, dos Reis R, Pereira J. Simulation of surface fire fronts using firelib and gpus. *Environmental Modelling & Software* 2012;38:167–177.

15. Singh B, Pardyjak E, Norgren A, Wilemsen P. Accelerating urban fast response lagrangian dispersion simulations using inexpensive graphics processor parallelism. *Environmental Modelling & Software* 2011;26:739–750.

16. Kalyanapu AJ, Shankar S, Pardyjak ER, Judi DR, Burian SJ. Assessment of gpu computational enhancement to a 2d flood model. *Environmental Modelling & Software* 2011;26:1009–1016.

17. Bryan BA. High-performance computing tools for the integrated assessment and modelling of socialecological systems. *Environmental Modelling & Software* 2013;39:295–303.

18. Timmermans J. Support to science element water cycle multi-mission observation strategy - design justification and validation. Tech. Rep. WACMOS-DJF-VR; ITC - University of Twente; 2011.

19. Timmermans J. Support to science element water cycle multi-mission observation strategy - primary analysis report. Tech. Rep. WACMOS-PAR; ITC - University of Twente; 2009.

20. Timmermans J. Support to science element water cycle multi-mission observation strategy - requirements baseline. Tech. Rep. WACMOS-RB; ITC - University of Twente; 2009.

21. Timmermans J. Support to science element water cycle multi-mission observation strategy - technical specification. Tech. Rep. WACMOS-TS; ITC - University of Twente; 2009.