iCeBOUND Project

Porting a Solar Energy Potential Algorithm on GPU

Supervisors: Pr Jan Hesthaven & Pr Nabil Abdennadher

Assistants: Dr Gilles Fourestey & Dr John White

Seif Ben Bader - MSc in Computational Science and Engineering

Problem Definition and Objectives

The project is about a solar energy potential application consisting in the assessment of the solar radiation in a town (we considered the city of Geneva, Switzerland as a use case). Given a **Digital Urban Surface Model** (here referred to as **DUSM**), the application aims at providing the user with qualitative information about the amount of solar radiation of every single location in the model according to a given sun position. The application acts as a Decision Support System for energy planification purposes, such as photovoltaic panels installation.

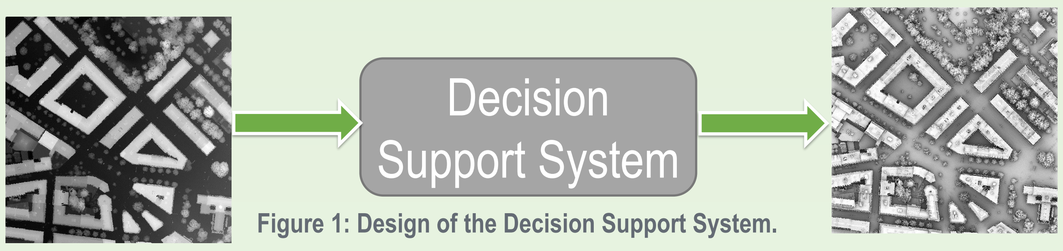

Figure 1: Design of the Decision Support System.

A whole cloud based benchmark of the solar energy potential application **(in Java)** already existed, but suffered from being **time consuming and expensive (in terms of cloud processing costs)**
This project reproduces from scratch a new **CUDA** (Compute Unified Device Architecture) version designed for GPUs, expected to tackle the previously cited drawbacks.

The Solar Radiation Model

The solar radiation is the sum of three components: the direct solar radiation, the diffuse solar radiation and the reflected solar radiation. These are the components involved in the calculation of the solar energy potential, and are illustrated in Figure 2.
The direct radiation is directly proportional to the sun visibility and whether a location is being shaded or not. The reflected radiation depends on the ground and the nearby object reflection. The diffuse radiation is derived from the sky visibility. The latter is the amount of visible portion of the sky and is given by the Sky Viewing Factor (referred to as SVF) which is a coefficient between 0 and 1; 0 being a totally obstructed point an 1 a point in a perfectly empty surrounding space. This is shown in Figure 3.

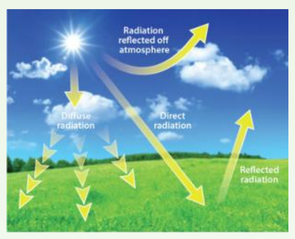

Figure 2: The solar radiation components.

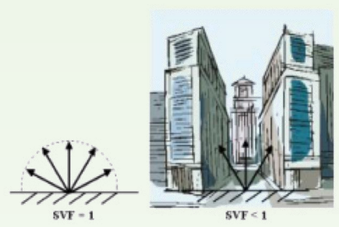

Figure 3: The Sky Viewing Factor illustration.

The amount of solar radiation for a point P denoted by $SR_l(P)$ for a given sun position l is given by the following formula: $SR_l(P) = I_{dir} * S_l(P) + I_{ref} + I_{dif} * h(SVF(P), S_l(P))$, where I_{dir}, I_{ref} and I_{dif} are three constants which depend on weather data, latitude, slope and orientation of the point P, whereas h is some function of $SVF(P)$ and $S_l(P)$, whom denote respectively the SVF and the shading state of the point P; 0 if shaded and 1 if sunny. The solar radiation model can therefore be resumed in the diagram Figure 4.

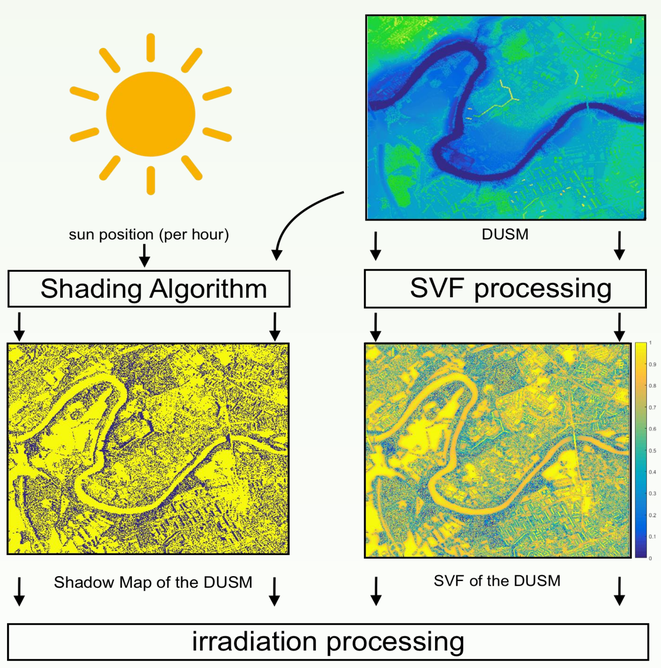

Figure 4: Block diagram of the irradiation process.

The Shading Algorithm

The Shading Algorithm renders a shadow map provided a DUSM and a sun position. For a given location in the DUSM, the algorithm tests every encountered neighboring point towards the sun direction and checks whether its height is bigger than the light beam source projection in which case the original point is shaded. In the opposite case, the original point is considered as still sunny and the algorithm moves on testing the next neighboring points. The algorithm stops if the point is shaded or a stopping criterion is encountered. This is shown in Figure 6 where P_0 is shaded by P_1 for sun position 2 and not for sun position 2. This algorithm is highly **irregular** and **time consuming**.

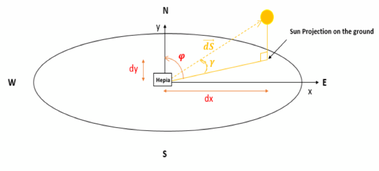

Figure 5: Algorithm follows the sun direction.

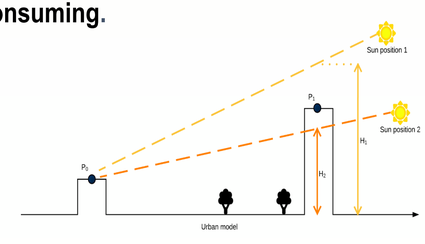

Figure 6: Illustation of the Shading Algorithm.

The SVF algorithm

The SVF is the percentage of sky that can be seen from a specific point. If we consider for instance that the sky is modeled by a cupola having different light sources uniformly placed on it, then the SVF can be assimilated to the ratio of light sources that enlightens the considered point. Therefore, the SVF of every point in an urban model can be calculated through an elegant use of the Shading Algorithm following the formula: $SVF(P) = \frac{1}{L} \sum_{l=1}^{L} S_l(P)$, where L is the number of light sources for modeling the sky. The number of light sources was set to $L = 400$. An SVF being therefore comparable to 400 shadow maps, it represents the heaviest and most important part of the irradiation process (c.f. Diagram in Figure 4)

Performance Analysis of the SVF process GPU version

The Geneva map was divided in 55 overlapping tiles, each of 3.4x3.4 km ^{2} and a half meter precision scale. Since the SVF process consists in a multiple Shading Algorithm execution which is highly irregular, the tiles were furthermore separated in tiles or Urban, Rural and Hybrid types. These are shown in Figure 7 (left), both with the SVF of the whole Geneva city (right). The average execution times corresponding to the new GPU implementation for DUSMs of different type are reported in Table 1. The Tesla K40 Nvidia GPU was used for the simulation.

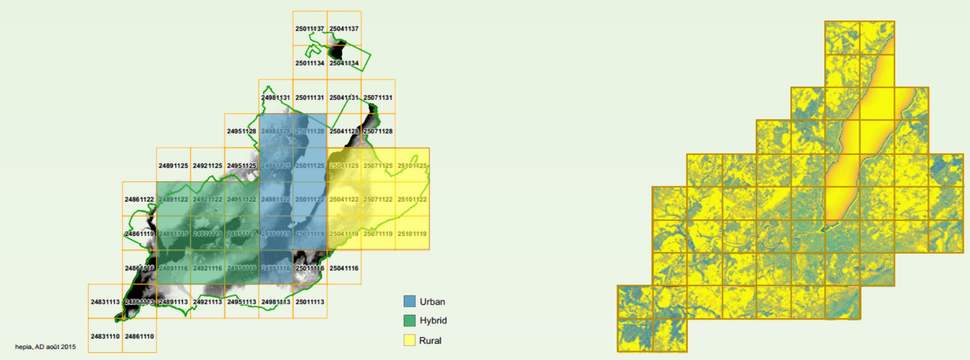

Figure 7: Map of the Geneva DUSM input tiles of different types (left part). The right part shows the SVF maps.

DUSM	Urban	Hybrid	Rural
Execution time	11m35	12m30	13m25

Table 1: Average execution times in **minutes** for DUSMs of different types.

SVF process on multiple GPUs

The Deneb cluster of EPFL was used for the purpose of executing the SVF process on multiple GPU devices. It is equipped with 16 GPU nodes containing 4 Nvidia Tesla K40 GPUs each. MPI was used for the purpose of allowing communication between nodes. The parallelization strategy consisted in distributing the light sources among multiple GPUs, as well as dividing the input tile (DUSM) in subparts for being processed by different GPUs. Figure 8 shows the diagram for a parallelization involving 8 GPU nodes. The whole tile is divided in four, and every 2 nodes (8 GPUs in total) are in charge of a sub-tile.

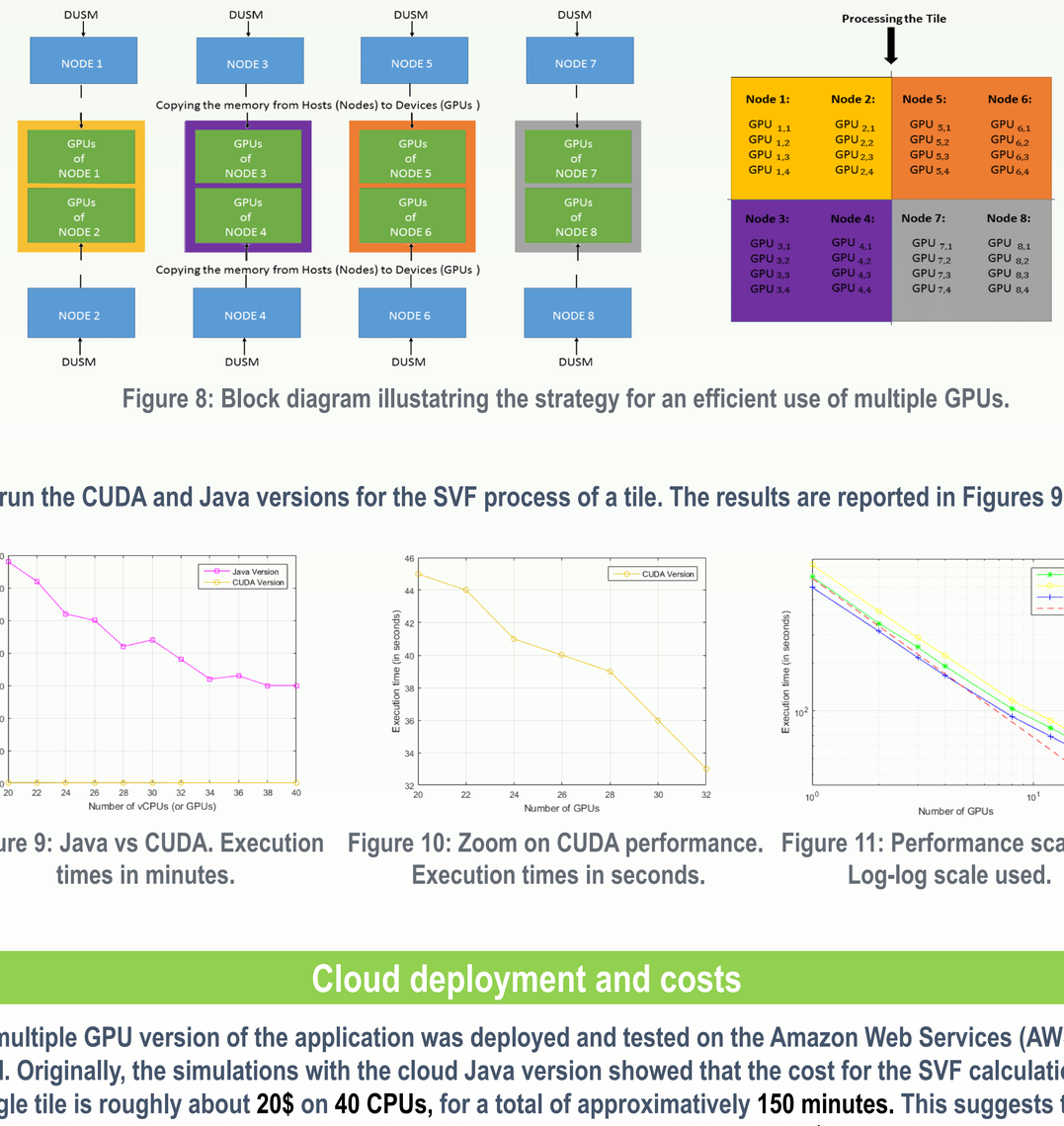

Figure 8: Block diagram illustatring the strategy for an efficient use of multiple GPUs.

We run the CUDA and Java versions for the SVF process of a tile. The results are reported in Figures 9,10,11.

Figure 9: Java vs CUDA. Execution times in minutes.

Figure 10: Zoom on CUDA performance. Execution times in seconds.

Figure 11: Performance scaling. Log-log scale used.

Cloud deployment and costs

The multiple GPU version of the application was deployed and tested on the Amazon Web Services (AWS) cloud. Originally, the simulations with the cloud Java version showed that the cost for the SVF calculation of a single tile is roughly about **20$** on **40 CPUs**, for a total of approximatively **150 minutes.** This suggests that the calculation of the SVF for the whole city of Geneva costs around **20x55=1100$** lasting for **5 days and 17 hours.**

With the new CUDA version, the SVF calculation of a single tile takes in average **16.5 minutes** on an instance containing a **single GPU** costs **0.65$.** When using an instance containing **four GPUs** (the maximum available), the cost of the calculation of the whole SVF is estimated to around **13$,** lasting approximatively **4.5 hours.**