GPU-based roofs’ solar potential estimation using LiDAR data

Niko Lukac\textsuperscript{a,1,2,*}, Borut \v{Z}alik\textsuperscript{a}

\textsuperscript{a}University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova ulica 17, SI-2000 Maribor, Slovenia

Abstract

Solar potential estimation using LiDAR data is an efficient approach for finding suitable roofs for photovoltaic systems’ installations. As the amount of LiDAR data increases, the non-parallel methods take considerable time to accurately estimate the solar potential. Although supercomputing provides a possible solution, it is still too expensive and thus infeasible for general usage. Fortunately, the recent graphics processing units (GPUs) can now be utilised to ensure fast computations. This paper proposes a novel method for fast solar potential estimation using GPU-based CUDA technology. This method employs LiDAR data, irradiance measurements, multiresolutional shadowing from solid objects, and heuristic shadowing from vegetation. Experimental results demonstrate the method’s effectiveness, in comparison with a multi-core CPU-based approach.

Keywords: Solar potential, LiDAR, GPU, CUDA.

1. Introduction

Renewable and clean energies are becoming extremely important due to the increasing costs of fossil fuels. As solar energy is a practically unlimited and free energy source, it has attracted a lot of attention over recent years, especially with more efficient photovoltaic (PV) systems being developed (Luque and Hegedus, 2003). In order to maximize energy production, various parameters have to be considered as those investors who install PV systems on buildings’ roofs need
to estimate the returns on their investments. Therefore, an efficient and accurate estimation of roofs’ solar potential is more than welcome. The major factor for the solar potential estimation of a surface is the solar irradiance (i.e. irradiation incident on a surface), which primarily depends on the roof’s local topographic properties, such as orientation and slope. Unfortunately, many other factors affect the irradiance: geographical location, shadowing, the impacts of atmospheric attenuation, and cloud cover. The local topographic properties can be accurately obtained using LiDAR (Light Detection And Ranging), an active remote sensing technology. A LiDAR scanner can obtain up to 200,000 points per second (Petrie and Toth, 2008), where the obtained point cloud consists of millions of non-structured points that have to be efficiently processed.

Nowadays, graphics processing units (GPU) can be used for intensive calculations, especially if the calculation tasks can be parallelized. This paper introduces a novel method for efficient solar potential estimation based on LiDAR data using general-purpose programming on GPU (GPGPU) with NVIDIA’s Compute Unified Device Architecture (CUDA) technology. The proposed method works on larger urban areas consisting of hundreds of buildings, and performs a local analysis on each building’s roof in order to estimate its solar potential. Several important location and time-dependant global influential factors are considered that affect the solar potential estimation. The site’s geographical location is obtained from the geo-referenced ALS (Airborne Laser Scanned) LiDAR data. Surface orientation (aspect) and its inclination (slope) are computed from the regular grid constructed from the LiDAR data. It is necessary that the direct and diffuse irradiances at a given location are estimated accurately. The best way is to utilise long-term measurements done with a pyranometer. Atmospheric attenuation (i.e. absorption and scattering) and cloud cover are dynamic meteorological factors that reduce direct irradiation, and cause diffuse irradiation. These factors are naturally included within the long-term diffuse irradiance measurements or approximated using an appropriate solar radiation model. Shadowing can significantly reduce the irradiances of the considered surfaces. The proposed method considers a multiresolutional shadowing approach: the hills surrounding an urban area are considered in lower resolution, whilst the urban area is treated in higher resolution. Shadowing from the high vegetation, as captured by LiDAR, is approximated by calculating the light extinction coefficients of the canopies, where the input is the Leaf Area Index (LAI) data. The spatiotemporal solar potential of the roofs’ surfaces is calculated as the average annual daily insolation, which is determined by an integral of the estimated solar irradiances throughout the day, with a given time step.
This paper is structured over 5 sections. Section 2 presents the used methods in Geographic Information Systems (GISs) for solar radiation modelling, the solar potential estimation using LiDAR, and the usage of parallel processing and GPGPU for geospatial analysis. Section 3 describes the proposed method in detail. The method’s effectiveness is demonstrated in Section 4. The last section concludes the paper.

2. Related work

Various methods have been developed for solar irradiance or radiation modelling, which have been implemented within various GIS software (Hetrick et al., 1993; Dubayah and Rich, 1995; Kumar et al., 1997; Fu and Rich, 2002; Corripio, 2003; Šúri and Hofierka, 2004; Pons and Ninyerola, 2008). These methods mostly operate on digital terrain models (DEMs) (Ruiz-Arias et al., 2009). However they are not specifically crafted for LiDAR data and do not consider multiresolutional shadowing or any influence from the vegetation. Šúri et al. (2007) estimated the PV potential (i.e. electricity generation) using a European solar map that was developed using the r.sun radiation model (Šúri and Hofierka, 2004). Similarly, Hofierka and Kaňuk (2009) used r.sun to assess the PV potential within urban areas. Wiginton et al. (2010) demonstrated methods for object detection, in order to determine suitable rooftops for PV systems’ installations.

Several works exist regarding solar potential estimation using LiDAR data. Voegtle et al. (2005) extracted the roof planes from the LiDAR data, then the suitable areas for PV systems were detected using a GIS database management system. Similarly, Kassner et al. (2008) masked the roofs’ contours, then a raster interpolation was performed, and the solar potential was estimated. Jochem et al. (2009a) introduced a more efficient buildings’ roof planes extraction, and performed solar potential estimation using the r.sun model. They considered shadowing effects of nearby objects by calculating the horizon for each point. The shadowing from the larger surrounding environment and cloud cover effects were already included within the meteorological data the authors used. Jochem et al. (2009b) introduced spatial variable vegetation shadowing by considering the first-last pulse of the laser data. Yu et al. (2009) investigated the spatiotemporal variation of solar radiation within an urban area derived from the LiDAR data. Levinson et al. (2009) estimated residential rooftops’ shading impacts on solar irradiance. They used a vegetation-growth model, where the properties of different vegetation were manually measured. Jochem et al. (2011) focused on extracting buildings’ vertical walls from Mobile Laser Scanning (MLS) data, and assessed
the walls’ solar potentials. Tooke et al. (2011) examined the temporal influence of
trees’ canopies on solar radiation within urban areas using LiDAR data. Nguyen
and Pearce (2012) developed a method for solar potential estimation that incorpo-
rates both terrain and near surface shadowing effects on LiDAR data.

With the increase in efficiency of computer hardware, many parallel methods
have been developed to speedup computationally demanding geospatial analysis.
Healey et al. (1997) demonstrated a case study of parallelizing a raster gener-
triangulation, which is suitable for faster solving of geometric problems in ter-
rain modelling. Han et al. (2009) proposed a parallel inverse distance weight-
ing (IDW) interpolation for creating DEM from ALS-based data. Krishnan et al.
(2010) considered the MapReduce parallel computing technology for local grid-
ding algorithms when generating DEM. Over the last few years there has been
major progress when using GPGPU for geospatial analysis. Walsh et al. (2009)
presented GPU-based implementations for three tasks in GIS: computational fluid
dynamics, seismic wave propagation, and rock magnetism. Ortega and Rueda
(2010) parallelized the drainage network computation from DEM using CUDA.
Beutel et al. (2010) parallelized the natural neighbour interpolation (NNI) algo-
rithm, based on Voronoi diagrams, in order to create DEM from massive point
clouds. Spatial processing using CUDA is also present in astrophysical data anal-
ysis (Jin et al., 2010). Oryspayev et al. (2012) proposed a GPU-based and multi-
core CPU-based method for LiDAR data vertices decimation, where the vertices
are represented by a triangulated irregular network (TIN). Steinbach and Ham-
nerling (2012) presented a GPU-based acceleration for raster operations during
the batch-processing of raster data in GIS. Tahmasebi et al. (2012) developed a
methodology that uses GPGPU and master-slave architecture for geostatistical
simulations based on random paths. Chen et al. (2012) performed 3D correlation
imaging for gravity on the GPU, by using CUDA.

Although there are some parallel methods for solar irradiance calculation in-
cluded within larger simulation models (e.g. Community Land Model (Oleson
et al., 2010), a part of Community Earth System Model), the proposed GPGPU
method is the first attempt for fast and accurate solar potential estimation using
LiDAR data. The accuracy is increased by considering more influential factors
such as multiresolution shadowing, heuristic shadowing from vegetation and irra-
diance measurements done with a pyranometer (Lukač et al., 2012). The proposed
method is described in detail in the next section.
3. GPU-based solar potential estimation

The proposed solar potential estimation GPGPU method is performed over three main steps: data preprocessing, shadowing, and estimation of the roofs’ solar potentials. The preprocessing of certain data is performed on the CPU before being used by the GPU for the computation of shadows and solar potential.

3.1. Data preprocessing

This method’s input is LiDAR point cloud that is classified into buildings, terrain, and vegetation classes with removed high and low outliers (see Fig. 1a). Many LiDAR data classification methods exist for extracting specific types of classes, for an overview see Jochem et al. (2009a); Heinzel and Koch (2011); Mongus and Žalik (2012). Each LiDAR point is arranged in the cell $C_j$ of a uniform rectangular grid data structure $\Gamma$ (see Fig. 1b). This data structure naturally supports the GPU-based computing, as the data from each cell can be processed independently using one streaming processor on the GPU. The resolution of the regular grid is user-defined according to the LiDAR point cloud resolution, similar to the virtual grid proposed by Han et al. (2009), but with different properties per cell. Each cell’s height ($C_{i,z}$) is equal to the highest point within a given cell.

Normally, the size of the cell is $1m^2$ or less, which is enough to preserve the topographic features of the roof planes. A given cell’s type is determined by the class of the points contained within the cell. In this way, a simplified 2.5D representation (i.e. 2D grid where cells’ values represent height) of the LiDAR point cloud is obtained. $\Gamma$ has an additional layer of boundary cells. These cells’ values are equal to the $boundary_{height} = \Gamma_{max\_height} + 1$, where $\Gamma_{max\_height}$ denotes the maximum height of the entire grid $\Gamma$. This enables fast out of grid’s boundary.
checking during shadowing, as explained in subsection 3.4. Besides the regular grid $\Gamma$ composed of terrain, buildings, and vegetation heights, the following data is needed:

- direct irradiances,
- diffuse irradiances,
- light’s extinction coefficients,
- daytime inverse sunbeams’ direction vectors.

This information is stored within separate data arrays. The sizes of the arrays for the average annual solar irradiance measurements and the light’s extinction coefficients’ are $365 \times 24 \times H$, where $H$ denotes the time interval factor relative to 1 hour (e.g. if the time interval is 30 min then $H = 2$). The light’s extinction coefficients determine the intensity of the shadowing from vegetation, and are described in Subsection 3.2. These are preprocessed on the CPU for an entire year, similar to the sunbeams’ direction vectors. The Solar Positional Algorithm (SPA) (Reda and Afshin, 2004) developed by the National Renewable Energy Laboratory (NREL) 3 is used for estimating the Sun’s position.

The sunbeams’ direction vectors are then calculated as $Sun_{pos} - \Gamma_{centre}$, where $\Gamma_{centre}$ represents the centre of $\Gamma$ and $Sun_{pos}$ the position of the Sun in Cartesian coordinates. Afterwards the direction vectors are normalized. During this pre-processing only the daytime sunbeams’ direction vectors are stored by checking whether the calculated position of the Sun is higher than the lowest cell within the grid. This considerably reduces the computational overhead, since there is no solar irradiance to be considered during night time. Preprocessing this data is efficient because it is re-used several times on the GPU. Before the shadowing and solar potential estimation can be performed, the normal vectors (i.e. vectors perpendicular to the cell’s surfaces) are calculated on the GPU for each cell in $\Gamma$. The cell’s normal $C_i.n$ for a given cell is obtained as an average of the normals from the neighbouring cells. The normals are vital for slope $\beta_i$ and orientation $\gamma_i$ angles extraction. $\beta_i$ is the angle between $C_i.n$ and the up vector of the horizontal plane, whilst $\gamma_i$ is the angle between the projected $C_i.n'$ on horizontal plane and geographical north, as shown in Fig. 2.

\footnote{3http://www.nrel.gov/}
3.2. Shadowing

Shadowing of the considered cell is checked by projecting solar rays’ (i.e. sunbeams) on the $\Gamma$ in order to test for the possible intersection of an object within a sunbeam’s path between a given cell and the Sun (see Fig. 3a). If an intersection occurs, the given cell is shadowed. The considered shadowing algorithm works similarly to that of a line rasterization algorithm, in the following steps:

1. Start at cell $C_i \in \Gamma$. The sunbeam projection is defined with vector $t$, which is initially set at $C_i$ ($t = (C_i.x, C_i.y, C_i.z) = (t.x, t.y, t.z)$).
2. Vector $t$ is incremented using the inverse sunbeam’ direction vector $l = (l.x, l.y, l.z)$ and the cell under $t$ is determined:
   (a) Incrementation: $t = t + l = (t.x + l.x, t.y + l.y, t.z + l.z)$
   (b) Cell determination: $C_j = \Gamma[\text{round}(t.x),\text{round}(t.y)], j \neq i \land C_j \in \Gamma$
3. If $t_z < C_j.z$, then $C_i$ is shadowed by $C_j$, and the algorithm returns to step 1, until all the cells have been tested. Otherwise rasterization continues in step 2, until the boundary of $\Gamma$ is reached.

The shadowed cells have a shadowing coefficient $S_i$ set either to 1 (shadowed) or 0 (not shadowed). Such a shadowing algorithm is computationally expensive. However, it is efficient for parallel implementation, where the cells can be tested independently and concurrently, if shadowed. The grid $\Gamma$ should be large enough
Figure 3: Illustration of the multiresolutional shadowing approach, where a) shows the shadowing performed within a high-resolution grid, and b) within a low-resolution grid. The shadowed cells are coloured blue, whilst the cells that cast shadows are coloured red. The inverse sunbeam’ direction vector is denoted as $L$.

to include all possible shadows casts by the objects and the surrounding environment (e.g. hills and mountains), until reaching the horizon. This approach requires large memory resources or an out-of-core solution. Therefore, this paper uses the spatiotemporal multiresolutional shadowing, where in addition to $\Gamma$, low-resolution data is used in the form of DEM stored within regular grid $\Sigma$. Essentially, $\Gamma \subset \Sigma$, where $\Sigma$ represents $\Gamma$’s surroundings. If a given cell $C_i \in \Sigma$ is shadowed then the high-resolution cells $C_j \in \Gamma$ covered by $C_i$, are also shadowed (see Fig. 3b). Shadowing within $\Sigma$ is computed faster, as the number of cells in $\Sigma$ is significantly lower than in $\Gamma$.

Vegetation shadowing is done in the same way as described above, by taking into account the Leaf Area Index (LAI), which supports variable vegetation shadowing throughout the different seasons of the year. The light extinction coefficient (Jackson and Palmer, 1979) based on LAI is used to define $S_i$:

$$S_i = 1 - e^{-K_{\text{LAI}}},$$

where $K \in [0, 1]$ is the extinction coefficient. $\text{LAI} \in [0, \infty]$ is defined as the ratio of the leaves within the canopy divided by the ground surface area. $K$ is estimated by the leaf inclination angle distribution of the trees’ canopies and zenith angles (Sinoquet et al., 2007), whilst many methods exist for LAI estimation (Zheng and
Moskal (2009), Yuan et al. (2011), Ganguly et al. (2012)).

3.3. Solar potential estimation

The proposed method considers the direct $I_b$ and diffuse $I_d$ irradiances. If several years of measurements from the pyranometer are available, the data is averaged during preprocessing. Else, solar radiation modelling can be considered (Ruiz-Arias et al., 2009). Reflective irradiance is not considered in this paper. For each cell $C_i \in \Gamma$, the instantaneous irradiance is calculated as:

$$I_i = I_b R_{b_i} (1 - S_i) + I_d R_{d_i} \left[ \frac{kW}{m^2} \right],$$

(2)

where the correction factor $R_{b_i} = \frac{\cos(\theta_i)}{\cos(\theta_{zi})}$ (Duffie and Beckman, 2006) is the ratio between the angles of incidence for a cell’s horizontal surface ($\theta_i$) and its tilted surface (i.e. zenith angle, $\theta_{zi}$) (see Fig. 2). The $R_b$ is required to address each cell’s tilting factor, because the $I_b$ measurements are typically made only for horizontal surfaces. For the diffuse irradiance, the correction factor is $R_{d_i} = \cos^2(\beta_i/2)$. The $\theta_i$ and $\theta_{zi}$ are defined as (Duffie and Beckman, 2006):

$$\cos(\theta_i) = \sin(\delta)\sin(\phi_i)\cos(\beta_i) -$$

$$- \sin(\delta)\cos(\phi_i)\cos(\beta_i) -$$

$$- \cos(\delta)\cos(\phi_i)\cos(\beta_i)\cos(\gamma_i) +$$

$$+ \cos(\delta)\cos(\phi_i)\sin(\beta_i)\cos(\gamma_i)\cos(\omega) +$$

$$+ \cos(\delta)\cos(\phi_i)\sin(\gamma_i)\sin(\omega),$$

(3)

$$\cos(\theta_{zi}) = \cos(\sigma)\cos(\phi_i)\cos(\omega) + \sin(\omega)\sin(\phi_i),$$

(4)

where $\phi_i$ is the cell’s latitude. Approximative solar declination is calculated as $\delta = 23.45^\circ \sin(360(284 + n)/365)$ (Cooper, 1969), where $n$ is the day in the year. The hour angle $\omega$ defines the location’s longitude $\lambda$ displacement relative to the Sun’s radiation. The $i$-th cell’s daily insolation is calculated by considering multiple instantaneous solar irradiances using a specific time step, from sunrise ($sr$) to sunset ($ss$):

$$J_i = \int_{sr}^{ss} I_i(t) dt \left[ \frac{kWh}{m^2} \right],$$

(5)

whilst the solar potential is simply defined as the annual average daily $J_i$:

$$P_i = \sum_{t=0}^{365} (J_i/365) \left[ \frac{kWh}{m^2} \right].$$

(6)
3.4. Implementation in CUDA

CUDA provides an efficient programming interface to address GPU as a parallel processor, in order to solve general-purpose problems (NVIDIA, 2012). CUDA-based GPGPU is much more robust and versatile in comparison with older shader-based GPGPU approaches. In the proposed GPU-based method, the CPU performs the preprocessing phase first, as described in Subsection 3.1. Then, the memory of the sizes of the input grids and preprocessed input data is allocated on the GPU. Afterwards, the input data is transferred from the host to the GPU’s allocated memory.

The CUDA logical memory layout is structured hierarchically, where individual processing threads are grouped in blocks, and these blocks are then grouped into grids, as shown in Fig. 4. At the hardware level, CUDA-enabled GPU consists of stream multiprocessors (SMs) that are further composed of stream processors (SPs) (i.e., CUDA cores). Each SM has low-latency memory available inside its own registers. A set of threads within the block can pass information through the shared memory. The global, texture and constant (i.e., read-only) on-device memories are accessible to all threads within the CUDA grid. The calculation is performed using procedural code (i.e., CUDA kernel) that is executed on each CUDA core. Therefore, GPU is considered a STMD (Single Thread Mul-

![Figure 4: Illustration of the proposed method's workflow, and CUDA's logical memory hierarchy.](image-url)
Figure 5: Illustration of combined heights input data grid $\Gamma$ converted to $\chi$ grid consisting only of building type of cells. The converted grid’s size is a multiply of $T$, in order to fully utilize the scheduling of the warps.

Multiple Data) system, as the same kernel is executed on different data. The proposed method handles each input grid’s cell on separate core (see Fig. 4). During the execution, the block of threads is divided into warps, where each warp typically consists of 16 or 32 threads (NVIDIA, 2012). Warps represent the core scheduling units to be executed on a SM. A warp can be delayed by a memory read instruction for several hundred cycles, and in the meantime another warp can be executed on the same SM. Hence, it is imperative to tile the threads in the most optimal way in order to increase occupancy. The number of threads per block is limited as well the number of threads per SM, which is GPU type specific. When considering the optimal amount of tiles, the SMs should have maximum occupancy. Block size is considered as $T \times T$ threads, and the CUDA’s grid size as $\left\lceil \frac{\Gamma_{\text{width}}}{T} \right\rceil \times \left\lceil \frac{\Gamma_{\text{height}}}{T} \right\rceil$ blocks, where $T$ defines the tile’s width.

It would be inefficient to calculate the solar potential on the entire $\Gamma$ grid, where multiple cells would have to be skipped, because they are not part of a building (e.g. terrain). Hence, threads that handle such cells would represent inefficient usage of resources. In order to solve this, the proposed method generates a second regular grid $\chi$ (see Fig. 5), consisting of only building type of cells, whilst also providing coalesced memory access. Each cell within $\chi$ has an index to its absolute position within $\Gamma$, in order to calculate normals and perform shadowing. At the beginning of the calculation, the index value is read from a given cell in $\chi$. Afterwards the cell in $\chi$ can be overwritten with calculated solar potential. The preprocessed light extinction coefficients ($\text{Light\_extinction\_coef}_{\text{ARRAY}}$) and irradiance data ($I_{b\text{ARRAY}}, I_{d\text{ARRAY}}$) are stored within the constant memory, whilst the input grids ($\Gamma, \Sigma$), daytime inverse sunbeams’ directional vectors ($\text{Sunbeams}_{\text{ARRAY}}$) and the input/output data grid ($\chi$) reside within the global memory. Shared memory is not utilized in the proposed approach. Hence it is preferable to configure CUDA to use more L1 cache than shared memory per SM.
Algorithm 1 CUDA kernel for solar potential calculation

Input: $\chi, \Gamma, \Sigma, \text{Sunbeams}_\text{ARRAY}, \text{Light}_\text{ext}_\text{coef}_\text{ARRAY}, I_{\text{ARRAY}}, I_{\text{ARRAY}}$

Output: $\chi$

1: $t_{id} \leftarrow \text{blockId}.x \ast T + \text{threadId}.x$
2: if $t_{id} \geq \text{num\_buildings\_cells}$ then
3:     return
4: end if
5: $C_i \leftarrow \Gamma[\chi[t_{id}]]$
6: $C_i.n \leftarrow \text{calc\_normal}(\text{find\_neighbour\_points}(\Gamma, C_i))$
7: $\beta_i \leftarrow \text{calc\_slope}(C_i.n)$
8: $\gamma_i \leftarrow \text{calc\_orientation}(C_i.n)$
9: $P_i \leftarrow 0$
10: for $h = 1 \rightarrow 365 \ast 24/H$ do
11:     $S_i \leftarrow \text{test\_if\_shadowed}(\Sigma, \text{Sunbeams}_\text{ARRAY}[h], C_i)$
12:     if $S_i = 0$ then
13:         $S_i \leftarrow \text{test\_if\_shadowed}(\Gamma, \text{Sunbeams}_\text{ARRAY}[h], \text{Light}_\text{ext}_\text{coef}_\text{ARRAY}[h], C_i)$
14:     end if
15:     if $S_i \neq 0$ then
16:         $\delta \leftarrow \text{calc\_declination\_angle}((h \ast H)/24)$
17:         $\omega \leftarrow \text{calc\_hour\_angle}((h \ast H) \bmod 24)$
18:         \[ \cos(\theta_i) \leftarrow \text{calc\_angle\_of\_incidence}(\delta, \omega, \phi_i, \beta_i, \gamma_i) \]
19:         \[ \cos(\theta_z) \leftarrow \text{calc\_zenith\_angle}(\delta, \omega, \phi_i) \]
20:         $P_i \leftarrow P_i + I_{\text{ARRAY}}[h] \ast (\cos(\theta_i)/\cos(\theta_z)) \ast (1 - S_i)/(365/H)$
21:     end if
22:     \[ P_i \leftarrow P_i + I_{\text{ARRAY}}[h] \ast \cos(\beta_i/2)^2/(365/H) \]
23: end for
24: $\chi[t_{id}] \leftarrow P_i$

$\Gamma$’s datatype is a structure of 2 floating-point values in order to store a solid object’s height (i.e. combined height of buildings and terrain), and the vegetation height. $\chi$ is represented as a 1D array of size equal to the number of buildings’ cells, where neighbouring cells belonging to the same building are stored consecutively. Fortunately, the solar potential calculation does not require any synchronisation between the cells, thus each thread performs stand-alone calculations for the assigned cell. The overview of the CUDA kernel is shown in Alg. 1. At first the thread index is calculated (line 1) and checked as to whether it is out of grid’s boundaries (line 2). This can occur because the $\chi$ size might not
always be divisible by tile width $T$. Hence the CUDA grid could be larger for 
$T - (\text{num\_buildings\_cells} mod T)$. Afterwards the cell $C_i$ within $\Gamma$ is read using 
index from $\chi$ (line 5).

**Algorithm 2** CUDA shadowing function pseudocode

**Input:** $\Gamma$, Sunbeam\_vector, Light\_ext\_coef, $C_i$

1: $h \leftarrow 0$
2: **while** $h < \text{boundary\_height}$ **do**
3: \hspace{1em} $C_i\.x \leftarrow C_i\.x + \text{Sunbeam\_vector\.x}$
4: \hspace{1em} $C_i\.y \leftarrow C_i\.y + \text{Sunbeam\_vector\.y}$
5: \hspace{1em} $C_i\.z \leftarrow C_i\.z + \text{Sunbeam\_vector\.z}$
6: \hspace{1em} $h \leftarrow \Gamma[\text{round}(C_i\.x) + \text{round}(C_i\.y) \ast \Gamma\_width]$
7: \hspace{1em} **if** $C_i\.z < h$ **then**
8: \hspace{2em} return(1)
9: \hspace{1em} **else**
10: \hspace{2em} **if** $C_i\.z < \Gamma[\text{round}(C_i\.x) + \text{round}(C_i\.y) \ast \Gamma\_width]$ **then**
11: \hspace{3em} return(Light\_ext\_coef)
12: \hspace{2em} **end if**
13: \hspace{1em} **end if**
14: **end while**
15: return(0)

Knowing $C_i$, it’s normal vector $C_i\.n$ can be calculated, using information from 
$C_i$’s neighbouring cells (line 6). Angles $\beta_i$ and $\theta_i$ are then computed using $C_i\.n$ 
(line 7-8). They only depend on the location and do not change throughout the 
considered time duration. The solar potential estimation is done with one loop 
for the whole year (line 10), where the number of iterations depend on specific 
time interval $H$. At first the multiresolutional shadowing approach is performed 
during the calculation time steps (line 11). If $C_i$ is not shadowed within the low-
resolution grid (i.e. shadowing coefficient $S_i = 0$) (line 12), then it is checked 
as to whether it is shadowed within the high-resolution grid (line 13), which can 
cause branch divergence. However, the computational cost of shadowing is sev-
eral times higher than the consequence of divergence. The shadowing method 
returns the shadowing coefficient $S_i$. The direct irradiance $I_b$ is considered only 
if $C_i$ is not shadowed or is partially exposed to the shadows (line 15). The thread 
calculates the necessary angles ($\sigma$, $\omega$, $\theta_i$, and $\theta_z$) (line 16-19) to estimate $I_b$ (line 
20). The diffuse irradiance is always considered (line 22), regardless of the shadow-
ning result. The result is stored within the $\chi$ grid (line 24). When the calculation
is complete, the results are copied from GPU back to the host memory.

The CUDA function for shadowing is shown in Alg. 2. Besides the heights grid and the cell to check if it is shadowed, the input requires inverse sunbeam’ direction vector and the light extinction coefficient. The shadowing rasterization algorithm iterates until the end of the grid, where the heights are equal with the boundary height. If a cell is shadowed by a solid object (i.e. building or terrain) (lines 6-7), then $S_i = 1$. Otherwise, it is checked whether it is shadowed by vegetation (lines 9-10). In the case of vegetation based shadowing, $S_i$ is equal to the light extinction coefficient (line 11). Shadowing within grid $\Sigma$ is performed the same way, except vegetation shadowing is not considered at such low-resolution.

4. Results

The proposed method using GPU and multi-core CPU approach was tested on two locations in Slovenia: 0.73km$^2$ part of Pekre town (46.549° N, 15.593° E) (see Fig. 6a), and 1.27km$^2$ part of Maribor city (46.559° N, 15.643° E) (see Fig. 6b). Both locations were provided in classified LiDAR data, and inserted into separate high-resolution grids $\Gamma$. The low-resolution grid $\Sigma$ was constructed using DEM and occupies 315km$^2$ of the surrounding area. The per cell resolution of $\Sigma$ was 25m $\times$ 25m. The direct and diffuse solar irradiance measurements were obtained using a pyranometer near the testing locations. The measurements were available for the last decade, and were averaged per-hour for a whole year. The vegetation in both areas is primarily deciduous, which resulted in more shadowing from vegetation during the summer and less in winter. The spatiotemporal LAI raster data was obtained from the Land-Atmosphere Interaction Research Group (Yuan et al., 2011), which used Moderate-Resolution Imaging Spectroradiometer (MODIS) technology for LAI estimation. The LAI measurements were available for 2001-2009 with resolution of 1km$^2$ per cell. In order to calculate light’s extinction coefficient, spatial LAI values were averaged over one year, and the $K$ was set at 0.75. Fig. 6 shows the calculated solar potentials for both tested areas using 1h time step and 1m$^2$ per cell resolution.
GPU calculation was done using NVIDIA Tesla C2050 GPU, which has 3GB video memory and 448 CUDA cores. Tesla’s peak single precision floating point performance is 1030 GFLOPS. The multi-core CPU calculation was done using two AMD Opteron 6274 CPUs (32 cores in total). All calculations were performed using single-precision, since the averaged solar irradiance measurements done using the pyranometer are stored in this way. The general speedup metric was considered for performance evaluation (Eager et al., 1989). Speedup ($SP$) defines the ratio between the time duration for $c$ CPU processors and $g$ GPU processors: $SP = T_c / T_g$. When using TESLA C2050 GPU, then $g = 448$, and for multi-core CPU $c = 32$. During the multi-core CPU approach, 64 concurrent threads gave the best results. The CUDA kernel was compiled with NVCC.

![Figure 6: Visualization of the tested areas (a) part of Pekre town and b) part of Maribor city) and the areas buildings’ solar potential. Buildings’ colours correspond to the intensity of the calculated solar potential, as shown in the gradient bar.](image)
(CUDA Toolkit 4.2) using -O2 option for speed optimisations. Similarly, the multi-core CPU code was compiled with GNU C using -O3 option.

Figure 7: Comparison of multi-core CPU and GPU runtimes when using a) different per cell resolutions with 1h time step and b) different time steps with 1m² per cell resolution.
Different grids’ per cell resolutions were tested in order to measure the spent computational time (see Fig. 7a) with constant 1h time step. Additionally, different time steps were considered (see Fig. 7b) with a constant per cell resolution of $1\,\text{m}^2$. The multi-core CPU times include only the times of the calculations, whilst the GPU times include the transfer times (GPU to host and vice versa), and the calculation times. The preprocessing time was, on average, below 3s and was excluded from the results, since both multi-core CPU and GPU use the same preprocessed data that was done on the CPU.

Figure 8: Comparison of runtimes on GPU using different CUDA block sizes on different per cell resolutions.
The CUDA block size was set at $32 \times 32$ to achieve the best performance. This was checked experimentally as shown in Fig. 8. The average solar potential for all the buildings within the area of Maribor was 1.766 kWh, and 1.658 kWh for part of Pekre town. The calculated solar potential on CPU was almost identical to the results from GPU. The solar potential slightly changed when the time step was decreased and the resolution increased, since the overall calculations were more accurate. This was true as long the grid resolution was not below original LiDAR point cloud spatial resolution, and the time step below the irradiance measurements temporal resolution. In Fig. 9 the speedup can be observed using GPU in comparison with the multi-core CPU approach. When considering different per cell resolutions for the Maribor and Pekre datasets, the average speedup was 20 and 14 times respectively. The average speedup when considering the different time steps for the Maribor and Pekre datasets was 28 and 31 times, respectively. In all cases the GPU approach was significantly faster than the multi-core CPU approach.

![Figure 9: GPU speedup ($SP$) in comparison with multi-core CPU when using different a) per cell resolutions with 1h time step and b) different time steps with 1m² per cell resolution.](image-url)
5. Conclusion

This paper proposed a new GPGPU method for solar potential estimation using LiDAR data, whilst considering several important influential factors that affect solar irradiance in order to achieve higher accuracy. Although these factors significantly increase the computational workload, as the amount of the input data increases, they can still be calculated on the GPU within a reasonable time. In comparison with the multi-core CPU, the GPGPU proposed approach is several times faster with the same degree of accuracy, as was demonstrated in the experiments. The overall accuracy of the solar potential estimation, mainly depends on the LiDAR data resolution, constructed grids resolutions, calculation time step, and the precision of the irradiance measurements.

Acknowledgements

We are thankful to the GEOIN company d.o.o (in Maribor; Slovenia) for providing the classified LiDAR datasets. Thanks to the Slovenian Environment Agency for providing the pyranometer measurements, and Land-Atmosphere Interaction Research Group for publicly sharing the LAI data. This work was supported by Slovenian Research Agency under grants P2-0041 and L2-3650.

References


