Implementation Strategy of NDVI Algorithm with Nvidia Thrust

Jesús Alvarez-Cedillo, Juan Herrera-Lozada, and Israel Rivera-Zarate

Instituto Politecnico Nacional
Parallel Processing Department
Up. Adolfo López Mateos Edif. CIDETEC, 07700 Mexico City
{jaalvarez,jlozada,irivera}@ipn.mx

Abstract. The calculation of Normalized Difference Vegetation Index (NDVI) has been studied in literature by multiple authors inside the remote sensing field and image processing field, however its application in large image files as satellite images restricts its use or need preprocessed phases to compensate for the large amount of resources needed or the processing time. This paper shown the implementation strategy to calculates NDVI for satellite images in RAW format, using the benefits of economic Supercomputing that were obtained by the video cards or Graphics Processing Units (GPU). Our algorithm outperforms other works developed in NVIDIA CUDA, the images used were provided by NASA and taken by Landsat 71 located on the Mexican coast, Ciudad del Carmen, Campeche.

1 Introduction

It is recognized that the degree of greenness, either during periods of drought or heavy rain can be an indicator of its strength and resilience to climate change conditions (Potter et al.). 1999, Cao et al. 2004; Stow et al. 2003, Peters et al. 2003). However, information on the tolerance threshold of the ecosystems of this region in years of extreme weather events does not exist. This paper calculates NDVI images for large images captured by satellite using the supercomputing as a tool and by using GPU video cards.

1.1 Normalized Difference Vegetation Index

It is well known in remote sensing that the relationship between bands near Infrared and red allow a verification test of the abundance or scarcity of vegetation in a region. The NDVI is used to identify the presence of green vegetation on the surface and characterize their spatial distribution and the state of evolution over time. This is determined primarily by weather conditions.

The interpretation of the index must also consider the phenomenological cycles and annual development to distinguish natural oscillations, vegetation changes in the temporal and spatial distribution caused by other factors. Water has reflectance $R > IRC$ [15][16] therefore NDVI negative values. Clouds have similar

R. Klette, M. Rivera, and S. Satoh (Eds.): PSIVT 2013, LNCS 8333, pp. 184–193, 2014.

values of R and CRF [16], so that its NDVI is close to 0. Bare soil with sparse vegetation has positive values although not very high. Dense vegetation that is moist and well developed presents the highest values of NDVI.

The NDVI has great value in ecological terms, as it is a good estimate of the fraction of photo synthetically active radiation intercepted by vegetation (FPAR) [1], primary productivity [2][3]. NDVI calculation exploits the properties of the high absorption bands in the visible and strong near infrared reflectance, with these values to find the relationship between the near infrared band (700-1300 nm) and red band (650 nm) which corresponds, in the TM, the relationship between bands 4 and 3 and SPOT, between strips 3 and 2. (TM4/TM3 or SPOT3/SPOT2).

The first use of vegetation index [4] was simply using the radiation of Infrared and red properties although this procedure is not named as vegetation index. The procedure shown was reported by Jordan et al, and still is useful, but has a big problem in that the range of values obtained can vary from 0 to infinity.

For this reason it is common to calculate NDVI (Normalized Difference Vegetation Index). NDVI was reported by Rouse et al. In 1973 [5], having the advantage that their values from -1 to +1. 1.2. EASY GPU PROGRAMMING

To design a small optimal program in the GPU, it is important to determine which options could be used for the programming.

Available options are:

1. CUDA: A set of native applications NVIDIA based on C language, is a powerful programming environment which requires experience to manage resources.
2. OpenCL: A set of native NVIDIA graphics applications based on OPENGL language.
3. THRUST: A set of optimized applications and simple based on C++ language: THRUST was selected because the optimized code is simple, having a small learning curve and works perfectly in any generation GPUs

THRUST is a template library for C++ and NVIDIA CUDA language, and based on the Standard Template Library (STL) [6]. THRUST can implement high-performance applications in parallel with a minimal programming effort through a high-level interface that is fully compatible with CUDA ,C and C++.

THRUST provides a collection of primitive parallel data, scanning data, sort information, and reduces operating expressions and formulas, which together can implement complex algorithms with simple and readable source code. To describe the calculations in terms of these high-level abstractions, this tool provides an option to develop efficient and optimal automatic applications. As a result, THRUST can be used to develop prototypes and CUDA applications, in terms of productivity, programming is very simple and concise, making it ideal where robustness and absolute return are very important. [7]

2 Previous Work

A wide range of NDVI changes having dispensed with these different models and apply large images knowing that its vegetation is abundant. This concept has been applied to SAVI (Soil Adjusted Vegetation Index) introduced by Huete (1988) [8]. The NDVI formula was used by Tucker & Sellers [9]. We also tested ARVI (Atmospherically Resistant Vegetation Index) reported by Kaufman and Tanre (1992) [10] for the EOS-MODIS. The ARVI changes NDVI equation behavior and its calculation is as follows:

Where, as before, NIR and RED (or VIS) is the response in the near-infrared and red (or visible) bands respectively.

Bearing in mind these principles, models have been applied to several areas of southern Mexico for a variety of crops, and natural vegetation. Three RGB bands, overall TM bands and 432 of the 321 Spot were used.

The blue band, is marked in the image areas as an vegetation index. Being a normalized index corrected the strong variations that exist in TM4/TM3 simple relationship between pixels of bands 4 and 3, which here is reduced to the limits of -1 to 1.

After applying the formula derived NDVI and once reaching the limits of negative and positive values, it is possible to indicate them in the image and, within their variants. The final phase is the map generation.

3 Implementation Strategy

3.1 Programming Model

THRUST operates two container types of vectors, host_vector & device_vector. As the name suggests, host_vector is stored in the memory of the CPU, while the device vector is processed in the GPU memory.

THRUST vector containers are defined as a typical output vector of C++, std :: vector. Similarly std :: vector, device_vector and host_vector are generic containers (capable of storing any type of data) that can be resized dynamically. Figure 1 shows the programing model of NVIDIA THRUST.

The equality operator can also be used to copy a host_vector to a vector or a host_vector to device_vector or device_vector to vector.

It is important to note that individual elements of a device_vector can be accessed using the standard bracket notation. However, for each of these accesses its necessary to call the function “cudaMemcpy” and this should be used sparingly.

3.2 Algorithm Development

Designing a parallel algorithm is complicated because there are no optimal tools to generate programs of this type. The development of such programs are scaled and there is a solution for every case which is more or less optimal.

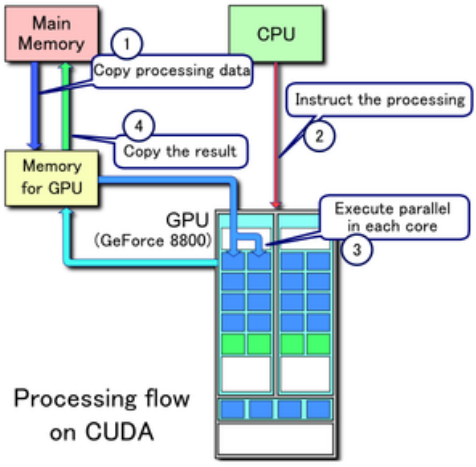

Fig. 1. Processing flow on NVIDIA CUDA and NVIDIA THRUST

To design the algorithm it was decided to generate a sequential process, as a base code and this source code was transformed to parallel code. Development requires the following concept formalization:

pixel where i=1,2,3,...n

As can be seen in contrast to the traditional method that takes a tour of the matrix in rows and columns, the suggested procedure only takes a tour of rows.

This feature is special, and indicates where to deploy and develop the device. This device manages memory vectors which consume raw or, simple memory, and this kind of memory is called linear.

The optimization model NVIDIA THRUST is an interface application to implement this style of programming without problems.

Trusth programming model has important differences to CUDA, to optimize the process, but using a lot of memory resources on the GPU. The low cost of the Nvidia card, allows run procedures directly, without having to partition the data problem. When the GPU memory is assigned, we create a transformation operator; this operator is a vector that executes in parallel the code inside, directly to architecture in a single step.

Thrust Nvidia handles transformation operators and direct operations in shared memory of the GPU; the operations it performs are called processing operations, and is typically found in the library file funcional.h as well as the basic operations. See Figure 2.

Our code is trying to solve NDVI, with a more simple,optimal code compared to the works reported by several authors in Literature with CUDA.

We defined our transformation operator source code as shown in Algorithm 1.

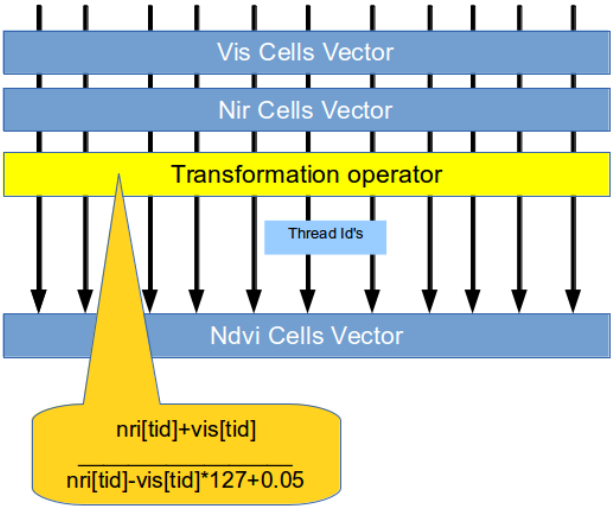

Fig. 2. Nvidia Thrust programming strategy

Algorithm 1. NDVI Cuda core source code

```
struct NDVI: THRUST::binary_function < char, char, char >
{
__device__float operator() (const char &A, const char &B )
{
return ( (( B + A ) / ( B - A ) ) +1 )* 127+0.05;
}
};
```

As can be seen, the transform operator is of a binary type, very similar in characteristic to CUDA, where the operator can acquire ownership of the device being executed at _device_, the CPU _host_ or both.

The typical computational complexity of this algorithm is $O(n_2)$, and after going through the incredibly convert operator in $O(k)$, where k is a constant number of instructions programmed into the transformation operator. (In our case a multiplication, division and a sum).Furthermore the manipulation of the image (read and write) is constant over time and can not be parallelized.Following the method and respecting the Thrust programming model:

Our proposed algorithm is shown in the algorithm 2.As we can see the programming style is simple and because it is a vector operation, this was designed in a simple style

The code #1 shows how the vector operation was defined and the final operation.

Algorithm 2. NDVI Paralell Algorithm

```
Input: Given a set of images I={I vis , I nir } R2 of n x n size
quantified in the range of an image [0,255] .

Output: one Indvi image ,2D , n x n size

1: load I vis to host_vis
2: load I nir to host_nir
3: Copy host_vis to device_vis
4: Copy host_nir to device_nir
5: Apply transformation operator to i item of NDVI in device_nir
6: Copy device_NDVI to I nir
```

4 Results and Analysis

4.1 Algorithm Validation

Among the great variety of urban spaces we only present an example of an image of positive values that indicates the method of analysis. But in each case study, small index variations result in new images of similar vegetation. To validate the algorithm, we compare our results with those generated with the tool ImageJ.

ImageJ is a Java image processing program inspired by NIH Image for Macintosh. It runs, either as an on-line applet or as a down loadable application.

Figure 3 and 4 show two gray-scale images, both images were obtained by Mex-sat, VIS and NIR respectively. Both images have a RAW format of 8547 rows x7585 columns at 8 bits.

typical behaviors.

Figure 5 shows the resulting NDVI image in (red-1.0) * (red-blue) at RAW format of 8547 rows x7585 columns at 8 bits and the corresponding histogram

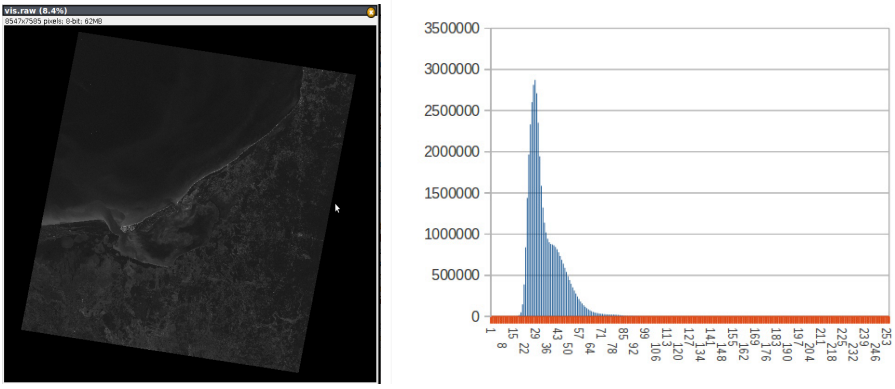

Fig. 3. Left:VIS image histogram right:Ciudad del Carmen, Campeche. Vis image in (red-1.0) * (red-blue) at RAW format of 8547 rows x7585 columns at 8 bits.

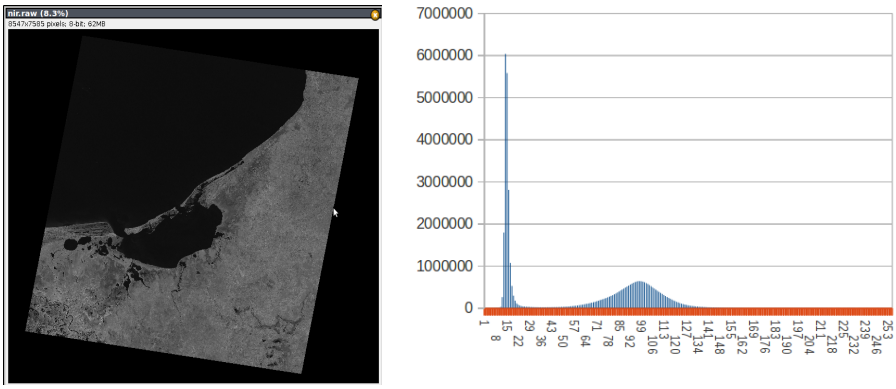

Fig. 4. left:NIR image, right:NIR image histogram

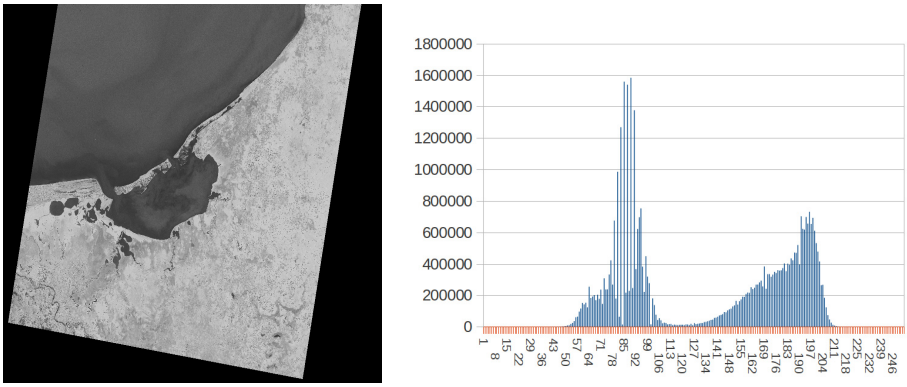

Fig. 5. Left:resulting NDVI image, right:Histogram resulting NDVI image

4.2 Stream Data

When analyzing data from the calculation of NDVI (Figure 6) it was observed that when the data is less than ten thousand, the sequential implementation is faster than any of the parallel implementations, but when the data is greater than ten million data, THRUST is faster than CUDA.

It can also be seen that in the parallel versions using linear vectors gives better speeds below ten thousand data, but when the data is greater than ten million times the best data was obtained using the floating type (primitive data).

This behavior is repeated with the use of images of a large amount of data (Figure 7- Top). When the data are greater than ten million times the best data was obtained when handling a floating rate. It is observed that the behavior of the execution times of the programs implemented with CUDA and THRUST have some degree of parallelism, THRUST being faster. This means that CUDA

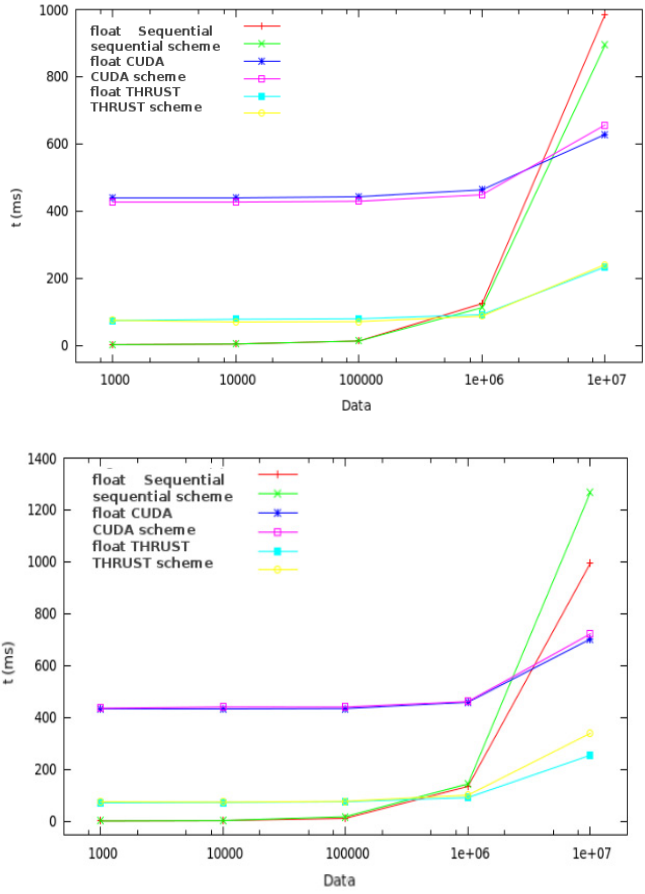

Fig. 6. Top:Comparison of different implementations with different types of NDVI, Bottom:Comparison of different implementations, with different types of large amounts of data

implementations used in this work are likely to be optimized while algorithms that use the interface of application (API) are optimized algorithms THRUST

In analyzing the performance of CUDA and THRUST versions regarding its sequential version, these are the best values of acceleration (speedup). Figure 7-Bottom was obtained using data from the floating rate rather than structures, and because algorithms are used THRUST optimized implementations, with THRUST running better than CUDA.

Similar behavior, where THRUST implementations require less time than those achieved with CUDA have been reported in other studies [11] [12][13][14].

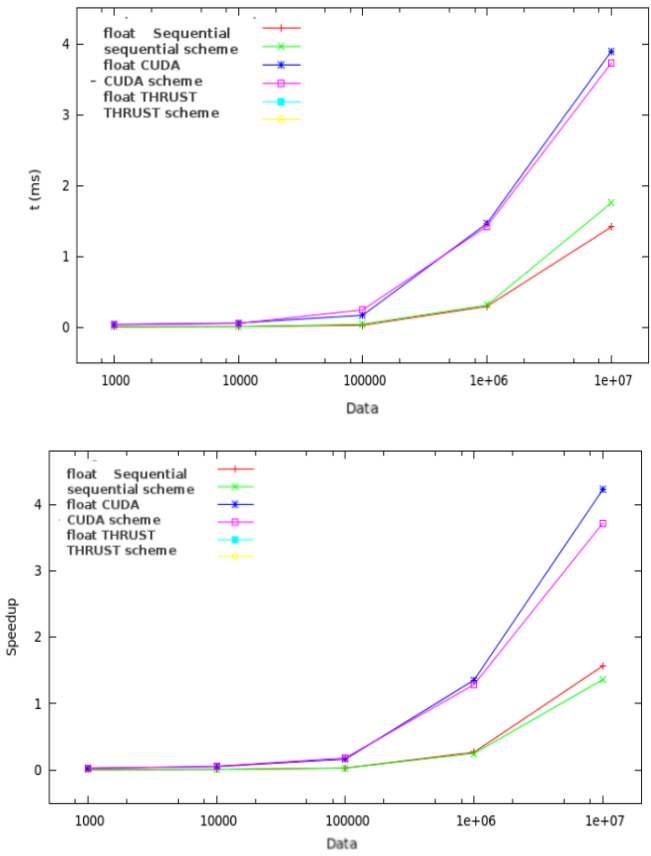

Fig. 7. Top:Comparison of speedups for CUDA and THRUST with different types of data when processed using NDVI , Bottom:Comparison of different implementations THRUST-CUDA

5 Conclusions

The parallelization of NDVI for a GPU can be implemented in CUDA or THRUST, however, as mentioned before, the use of CUDA means having knowledge of architecture at the hardware level. Such work must be distributed among the GPU processors through the allocation of threads, blocks, grids. On the other hand when using THRUST programming this does not require knowledge; in addition this programming is more efficient, but this does not imply that it is possible to achieve these levels of efficiency with CUDA and finally THRUST is a library based on CUDA.

Note that the resulting code using THRUST is less than is obtained when using CUDA, as well as its interface is similar to that of the STL (Standard

Template Library) which makes implementation simple, and can be combined in a single CUDA program.

Regarding the type of data, the best implementations were achieved when using float type data structures instead of generalizing on the basis of this work.

A better implementation of this algorithm is based on primitive data instead of complex data.

References

1. Zhang, Y., Tian, Y., Knyazikhin, Y., Martonchik, J.V., Diner, D.J., Leroy, M., Myneni, R.B.: Prototyping of MISR LAI and FPAR Algorithm with POLDER Data over Africa. IEEE Transactions on Geoscience and Remote Sensing 38(5) (2005)
2. Paruelo, J.M., Epstein, H.E., Lauenroth, W.K., Burke, I.C.: ANPP estimates from NDVI for the central grasslands region of the U.S. Ecology, 953–958 (1997)
3. Tucker, C.J.: Red and photographic infrared linear combinations for monitoring vegetation. Rem. Sens. of Environ. 8, 127–150 (1979)
4. Jordan, C.F.: Derivation of leaf area index from quality of light on the forest floor. Ecology 50, 663–666 (1969)
5. Rouse, J.W., Haas, R.H., Schell, J.A., Deering, D.W.: Monitoring vegetation system in the great plains with ERTS. In: Ecology Third ERST Symposium, NASA SP-351, vol. 1, pp. 309–317 (1973)
6. NVIDIA Co., CUDA Toolkit 4.0, THRUST Quick Start Guide, PG-05688-040_v01 (2011)
7. Ruestch, G., Micikevicius, P.: Optimizing Matrix Transpose in CUDA. Tech report, NVIDIA (2009)
8. Huete, A.R.: A Soil-Adjusted Vegetation Index (SAVI). Remote Sensing of Environment 25, 295–309 (1988)
9. Tucker, C.J., Sellers, P.J.: Satellite remote-sensing of primary production. International Journal of Remote Sensing, 1395–1416 (1986)
10. Kaufman, Y.J., Tanre, D.: Atmosoherically resistant vegetation index (ARVI) for EOS-MODIS. In: Proc. IEEE Int. Geosci. And Remote Sensing Symp. 1992, pp. 261–270. IEEE, New York (1992)
11. Faber, R.: Cuda Application Design and Development. Elsevier (2011)
12. Xiu, D.: Numerical Methods for Stochastic Computations: A Spectral Method Approach. Princeton University Press (2010)
13. Rubinstein, R.Y.: Simulation and the Monte Carlo Method. John Willey and Sons (1981)
14. Rosenthal, J.S.: Parallel computing and Monte Carlo algorithms. Far East Journal of Theoretical Statistics 4, 207–236 (2000)
15. A.S Hope, Estimation of wheat canopy resistance using combined remotely sensed spectral reflectance and thermal observations. In: Department of Geography, San Diego State University, San Diego, California 92182 USA (2010), http://dx.doi.org/10.1016/0034-4257(88)90035-1\
16. King, M.D., Kaufman, Y.J., Menzel, W.P., Tanre, D.: Remote sensing of cloud, aerosol, and water vapor properties from the moderate resolution imaging spectrometer (MODIS), Geoscience and Remote Sensing. IEEE Transactions Geoscience and Remote Sensing 30(1), 2–27 (1992)