Davor SLUGA, Tomaž CURK, Blaž ZUPAN, Uroš LOTRIČ

Faculty of Computer and Information Science, University of Ljubljana, Slovenia

Acceleration of information-theoretic data analysis with graphics processing units

Abstract. *Information-theoretic measures are frequently employed to assess the degree of feature interactions when mining attribute-value data sets. For large data sets, obtaining these measures quickly poses an unmanageable computational burden. In this work we examine the applicability of consumer graphics processing units supporting CUDA architecture to speed-up the computation of information-theoretic measures. Our implementation was tested on a variety of data sets, and compared with the performance of sequential algorithms running on the central processing unit.*

Streszczenie. *Miary informacji takie jak informacja wzajemna są często używane do określania stopnia współzależności cech podczas eksploracji zbiorów danych opisanych atrybutami. Dla dużych zbiorów danych, proste wyliczanie tych miar prowadzi wprost do znacznego wzrostu nakładów obliczeniowych. Praca jest poświęcona możliwości zastosowania programowalnych kart graficznych do przyspieszenia wyznaczania miar informacji. Nasza implementacja została przetestowana na różnych zbiorów danych oraz porównana z implementacją sekwencyjną na procesorze głównym.* ***(Przyspieszenie analizy danych opartej o teorię informacji przy pomocy programowalnych kart graficznych)***

Keywords: information-theoretic measures, data mining, parallelization, CUDA
Słowa kluczowe: miary informacji, eksploracja danych, algorytmy równoległe, CUDA

Introduction

In many fields of engineering and research constant improvements in data acquisition techniques lead to the production of large amounts of experimental observational data. However, analysis and modelling of very large data sets, especially in the case when the number of observed attributes (features, variables) is large, quickly poses an unbridgeable obstacle for current algorithms and computer systems.

To ease the task of analysis of data with large number of features, feature reduction methods can be used to remove less important attributes. They simplify the model inference by reducing the computer memory requirements on one side and potentially improving the generalization capabilities on the other side.

Many feature reduction methods compare the linear relationship of two random variables, e.g. an attribute and a class, and thus suggest a set of attributes which effectively represent the contained information. Yet there exist methods that compare the non-linear relationship of multiple random variables, i.e. a set of attributes and a class [1]. A class of such methods is based on the information-theoretic measures originating from Shannon entropy [2]. The information-theoretic methods use second-order statistics which becomes extremely important when large data sets are in question [3].

Unfortunately, the computation of information-theoretic measures is a complex operation leading to time-consuming feature reduction, especially in cases of data sets with large number of attributes. However, the data parallelism inherently present in the proposed feature reduction method makes it perfect candidate for processing on modern graphics processing units.

Commodity computer graphics processing units (GPUs) are probably today's most powerful computational hardware available at an affordable price. From the hardware that was specifically targeted to speed-up the 3D computer graphics rendering, they have evolved into a platform that nowadays supports general-purpose computing [4]. These computing components offer a massively parallel architecture with high memory bandwidth and substantially more computing power than offered by central processing units (CPUs). They are now frequently included in commodity laptop and desktop computers. The research on using GPUs for general-purpose computing has been ongoing for about two decades, but the real breakthrough occurred in 2007 when Nvidia, the leading GPU vendor, released a proprietary development platform known as the compute unified device architecture (CUDA). CUDA allows for general purpose programming of Nvidia's consumer graphics hardware in a language practically identical to C but with a few extensions. This has spawned a surge of interest in exploiting the GPUs for the scientific computation in a broad area of scientific fields ranging from engineering [5, 6] to natural sciences [7], and bioinformatics [8, 9]. The related programming tools are today improving rapidly and the GPUs are becoming more and more powerful. Some of them, like the Nvidia Tesla series of GPUs, are even specifically targeted at general-purpose computing [10].

The remainder of the paper is organized as follows. In the next section the information-theoretic measures implemented in our data analysis toolbox are described. Next, we briefly present the concept of the GPU in terms of CUDA platform. We continue with the implementation details of the selected information-theoretic algorithms for the CUDA platform are presented. We have tested the performance of the proposed solutions on various attribute-value data sets of different sizes. The performance figures of the GPU implementation are given in comparison to the CPU implementation. We conclude the paper with a summary of the main findings on the applicability of the GPUs for information-based data analysis.

Information-theoretic measures

A frequent task in machine learning is inference of a function from the training data. In the supervised learning one tries to build a model which relates a set of input random variables (attributes) to an output random variable (a class). Techniques based on the information theory enable ranking of the attributes $A^1, A^2, \ldots, A^N$ according to the average information they provide about the value of the class C. This can be quantitatively measured using Shannon's measures of information entropy and mutual information [11].

Given the class C, in which the class labels $c_1, c_2, \ldots$ appear with the probabilities $p(c_1), p(c_2), \ldots$, its uncertainty in terms of information entropy becomes

$$H(C) = -\sum_i p(c_i) \log_2 p(c_i) \,. \tag{1}$$

Suppose that the class labels depend on the attribute A^k, which takes the discrete values $a_1^k, a_2^k, \ldots$ with the probabilities $p(a_1^k), p(a_2^k), \ldots$. Then, the conditional entropy of the class C given the particular attribute value $A^k = a_j^k$ can be calculated as

$$(2)\quad H(C|A^k = a_j^k) = -\sum_i p(c_i|a_j^k) \log_2 p(c_i|a_j^k)\ ,$$

where $p(c_i|a_j^k)$ is the probability of obtaining class label c_i given the attribute value a_j^k. So, the uncertainty of the class C given the attribute A^k is measured by the conditional entropy

$$(3)\qquad H(C|A^k) = \sum_j p(a_j^k) H(C|A^k = a_j^k)\ ,$$

and finally, the amount of information that the attribute A^k provides about the class C is given by the bivariate mutual information

$$(4)\qquad I(C; A^k) = H(C) - H(C|A^k)\ .$$

The mutual information is zero when the attribute and the class are independent of each other, i.e. $p(c_i|a_j^k) = p(c_i)$, and positive otherwise.

The multivariate non-linear relationship among three or more attributes can be revealed by extending the concept of mutual information as described above. One such approach was proposed in [1], which proposes an extension of Eqs. 2, 3, and 4. To determine the amount of information that the pair of attributes A^m and A^n, $m \neq n$, provides about the class C, their Cartesian product needs to replace the attribute A^k in the above equations, i.e. $A^k = A^m \times A^n$. It is important to note that the three-variate mutual information $I(C; A^m \times A^n)$ obtained in this way, significantly differs from the mutual information common to three random variables $I(C; A^m; A^n)$ defined in [2]. However, the latter can be derived from the former by subtracting the mutual information of the class and each individual attribute, $I(C; A^m)$ and $I(C; A^n)$ [2].

To obtain a set of the most informative attributes, all possible interactions should be calculated and then only the attributes above a certain threshold retained. While the bivariate mutual information is calculated for each of N attributes and the class, the three-variate mutual information is calculated for each of $N(N-1)/2$ pairs of attributes and a class when the symmetry in attributes is considered. If the number of attributes is large, then solving the problem apparently becomes computationally very expensive.

GPU and CUDA

GPUs combine numerous processing cores on one chip. For instance, a modern GPU Nvidia Tesla C2050 has 14 processing units, called multiprocessors, each of them combining 32 computational units called CUDA cores. GPUs dedicate much more hardware per core to data processing than CPUs, but lack sophisticated control units [12]. This makes GPUs ideal for problems in which the same program is executed on many data elements in parallel.

From the programmers point of view the GPU basically acts as a powerful parallel coprocessor that can be used to speed-up parts of the algorithm, which exhibit enough data parallelism. The programmer has to identify the parallelism in the algorithm and divide the processing into many small, independent subtasks, that can be run in parallel. The execution of such program (see Fig. 1) begins on the CPU (referred to as the host) and is executed serially until a GPU (referred to as the device) section of the program is encountered. If there is any data needed by the device that resides in the host's main memory, the data is copied to the device memory (referred to as the global memory) first. The GPU section of the program is then executed on the device in parallel by the numerous threads. After the GPU has completed the execution of it's part, the output data is transferred back to the host's main memory for further processing, if needed.

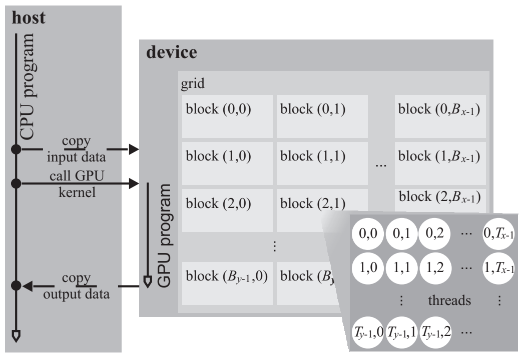

Fig. 1. CUDA execution model. Blocks and threads are identified with two-dimensional indices

The device code is executed in parallel by many independent threads, which are organized into blocks of threads that are joined in a grid of blocks. All of the threads of a grid execute the same code (referred to as kernel). However, the programmer has the mechanisms to differentiate between thread blocks and threads and can thus branch the execution depending on the current thread. The block and the threads can be addressed with one- or two-dimensional indices, depending on the problem domain.

Each thread has it's own set of registers and is also able to access the so called shared memory and the global memory. The shared memory resides in the multiprocessor and is common to all threads of a block. This memory acts as a fast cache, but it is different from the normal CPU cache because the programmer can explicitly define which data should be kept in the shared memory. However, recent GPUs, namely the Fermi architecture from Nvidia, enables the programmer to use a part of the shared memory as an implicit cache [12].

When writing efficient code for the GPU, the programmer must follow certain rules [10]. First, to utilize all of the processing resources of a GPU, there should be enough threads executing at any given time. As different Nvidia GPUs have different number of multiprocessors, CUDA cores, and other resources, and as the requirements per thread depend on the algorithm, this number can vary considerably. Second, the number of divergent branches inside a thread block should be kept at minimum, otherwise the code is forced to execute serially for each branch taken by different thread. Third, the register and shared memory usage should be maximized since the access latency to this two types of memories is very small in comparison with the global memory. Next, the data transfer between GPU memory and the CPU main memory should be kept as low as possible since this is certainly a bottleneck situation. Finally, the memory access pattern must be taken into account. The time required to access the global memory can be reduced, if threads request a single continuous segment of the global memory. Neglecting these rules can seriously cripple the algorithm's performance.

Computation of the measures on a GPU

In data sets with large number of attributes, the computation of the mutual information can be computationally extremely expensive. Luckily, when bivariate relationships are considered, the computation of the mutual information

$I(C;A^m)$ and the mutual information $I(C;A^n)$, $m \neq n$, is data-independent. Consequently, we can design effective algorithms for their computation on the GPU architecture. Using CUDA, we have to partition the problem into tasks that can be efficiently mapped to the GPU threads. The execution of the algorithm, as shown in Fig. 2, is split into three phases.

```
read data from file
evaluate probabilities $p(c_i)$
calculate $H(C)$
send $H(C)$ and data to GPU
for each attribute $A^k$ do in parallel
  evaluate probabilities $p(a_j^k)$
  for each $a_j^k \in A^k$ do
    evaluate probabilities $p(c_i|a_j^k)$
    calculate $H(C|A^k = a_j^k)$
  calculate $I(C;A^k)$
retrieve $I(C;A^k)$ for all $A^k$ from GPU
return attributes $A^k$ with suitable
$I(C;A^k)$
```

Fig. 2. Pseudo-code of the parallel algorithm. The code written in bold is executed on the GPU

The first phase starts on the CPU by reading data and calculating entropy $H(C)$ according to Eq. 1, where probabilities $p(c_i)$ are evaluated by counting occurrences of each class label c_i. The rather quick computation of $H(C)$ is performed on the CPU to avoid additional overhead caused by invocation of an extra kernel call. After sending relevant data the computation continues on the GPU.

In the second phase, the parallel computation is performed on the GPU, where for each attribute A^k a thread is invoked in order to calculate the mutual information $I(C;A^k)$. The GPU threads are identified by means of a block index i^B and an index of a thread i^T inside a block. We obtained the best performance by empirically setting the number of threads in a block to $B = 256$ and consequently using $\lceil N/256 \rceil$ thread blocks. Thus, the thread identified by indices i^B and i^T is responsible for computations regarding the attribute A^k, where $k = Bi^B + i^T$. Next, the probabilities $p(a_j^k)$ of attribute values are obtained similarly as in the case of class labels. Then, sequentially, for each given attribute value a_j^k the conditional probabilities $p(c_i|a_j^k)$ are evaluated by counting occurrences of class label c_i. For these $H(C|A^k = a_j^k)$ is assessed according to Eq. 2. Further on the mutual information $I(C;A^k)$ is obtained by using Eqs. 3 and 4.

Finally, when the execution on the device has finished, we simply copy the resulting data into the computer main memory. Afterwards, the obtained results can be filtered according to the given criteria in order to retain only the relevant attributes for.

When the three-variate relationships are considered by the mutual information $I(C;A^m \times A^n)$, the above algorithm needs to be slightly modified. Each attribute A^k now becomes a combination of two original attributes, $A^k = A^m \times A^n$. In this case, the number of attribute values dramatically increases to $|A^k| = |A^m| \cdot |A^n|$ where $|\cdot|$ denotes the number of different values of an original attribute.

Since the number of composed attributes and consequently the number of parallel threads increases to $N(N-1)/2$, the one-dimensional indexing of the threads cannot be used. Namely, for N around ten thousand, the upper bound on the number of parallel threads posed by the hardware limitations is reached. Using threads organized in two-dimensional arrangement is suitable for many practical applications. However, in our situation this poses additional problems. Each thread is identified by a two dimensional block index (i_x^B, i_y^B) and a two dimensional index of a thread inside a block (i_x^T, i_y^T) (see Fig. 1). From those indices a linear index can be obtained $i_L = (B_y i_y^B + i_x^B)T_x T_y + (T_y i_y^T + i_x^T)$, with B_x, B_y, T_x, T_y being the numbers of blocks and threads in given dimensions. The index i_L is then used to identify the two original attributes A^m and A^n needed by a thread,

$$(5)\quad m = \left\lfloor \frac{1+\sqrt{1+8i_L}}{2} \right\rfloor \ ,\ n = i_L - \frac{m(m-1)}{2}\ .$$

Because of the limited precision of GPU floating point operations, namely the square root function provided by the standard CUDA math library, a special integer square root function was implemented to avoid indexing errors in Eq. 5 [13].

Experimental work

To asses the performance of the GPU implementation of the algorithm for calculation of mutual information, a set of experiments was performed on an assortment of data sets. The GPU implementation was compared to a sequential version written in C language that was executed solely on the CPU.

All data sets were artificially generated by varying the number of attributes as well as the number of samples. Without the loss of generality two-valued (binary) attributes and classes were used in all data sets. All tests were performed on a computer with Intel Core 2 Duo E8400 3 GHz CPU, 4 GB of the main memory, and the Nvidia Tesla C2050 GPU with 3 GB of the global memory. The measurements of execution times of both implementations were performed ten times to assure relevant results. The average execution time of the GPU implementation t_{GPU} and the average time of the CPU implementation t_{CPU} were used to determine the average speed-up, $Speedup = t_{\mathrm{CPU}}/t_{\mathrm{GPU}}$.

Performance of the proposed algorithm for computing the bivariate relationships was first evaluated by changing the number of samples from 1000 to 10000 in steps of 1000 with the number of attributes kept constant at 5000. The execution times and speed-ups with standard deviations are presented in Fig. 3.

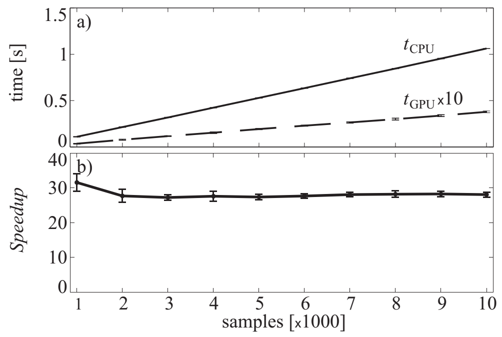

Fig. 3. Processing times a) and speed-ups b) with respect to the number of samples when computing bivariate relationships

To compare the execution times in the same chart, the t_{GPU} is multiplied by 10. As expected, the execution times in both cases increase proportionally with the number of samples. The speed-ups of approximately 30 are in favour of the GPU implementation, however the parallelisation cost pays off only when datasets with very large number of attributes are considered.

In the next experiment the number of samples was kept constant at 1000 while the number of attributes was changing from 1000 to 10000 in steps of 1000. As shown in Fig. 4, the processing times of both implementations increase linearly with the number of attributes. However, in the case of the GPU implementation the slope slightly changes at approximately 3000 attributes. This is due to the fact that the GPU is not fully occupied for data sets with small number of attributes. It is even more evident on the speed-up chart, where the speed-ups become roughly constant only when data sets include more than 4000 attributes.

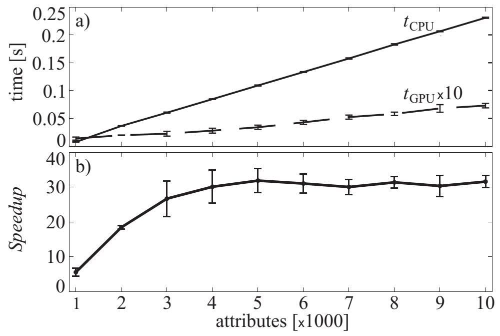

Fig. 4. Processing times a) and speed-ups b) with respect to the number of attributes when computing bivariate relationships

Performance of the proposed algorithm for computing the three-variate relationships was evaluated in the same way using the same data sets. Whilst the dependence on the number of samples demonstrates similar behaviour as above, the execution times with respect to the number of attributes increase quadratically for both implementations. The execution times can be seen in Fig. 5, where t_{GPU} is multiplied by 100 to put it on the same scale. The speed-ups shown in Fig. 5 are nearly constant for data sets with large number of attributes. They are much higher than in the bivariate case, which indicates that the computing of the three-variate relationships is far more complex in comparison with the bivariate ones. Higher speed-ups, obtained for the data sets with larger number of attributes, can be associated with the CPU cache size. More precisely, in the case of 1000 attributes the whole data set fits into the CPU cache, while in all other cases the cache misses occur. Large execution times imply the suitableness of the GPU for computing of the three-variate relationships.

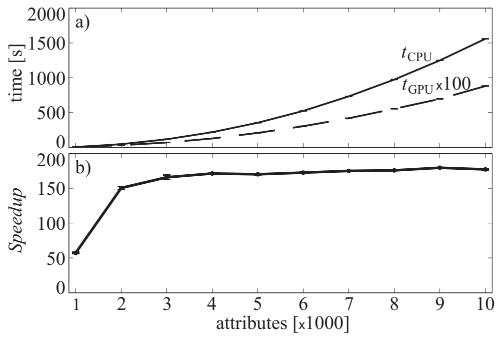

Fig. 5. Processing times a) and speed-ups b) with respect to the number of attributes when computing three-variate relationships

In the same manner the relationships among four or even more attributes and a class could be calculated. In such cases, the GPU would be fully occupied at even smaller number of attributes on one side, but on the other side more complicated mappings of attributes to threads would probably result in smaller speed-ups.

Conclusion

We have observed a tremendous performance when estimating bivariate and three-variate relationships among attributes and the class from attribute-value data sets using the computation of information-theoretic measures on the graphical processing units. This can be ascribed to the very high level of parallelism inherent to the problem. The speed-ups we achieved show that the GPU's resources were utilized very efficiently which is what we seek when parallelising algorithms for the GPUs.

The test data sets included unusually large number of attributes for conventional data mining. However, in many modern data analysis tasks, like in bioinformatics, market basket-analysis and recommendation systems data sets of this and larger sizes abound [14]. Besides feature reduction, data mining may have additional interests in discovery of synergistic interactions [15] of which estimation is based on computation of information-theoretic measures. These may be yet another possible application of the architecture and algorithms that we have proposed in our work.

BIBLIOGRAPHY

[1] Hild II K.E., Erdogmus D., Torkkola K., Principe J.C.: Feature Extraction Using Information-Theoretic Learning, IEEE Transactions on Pattern Analysis and Machine Intelligence, 28 (2006), No. 9, 1385–1392.
[2] Cover T., Thomas J.: Elements of Information Theory, New York, Wiley (2006).
[3] Lee C., Landgrebe, D.: Feature Extraction and Classification Algorithms for High Dimensional Data, TR-EE 93-1, School of Electrical Engineering, Purdue University, Indiana (1993).
[4] Owens J.D., Houston M., Luebke D., Green S., Stone J.E., Phillips J.C.: GPU Computing, Proceedings of the IEEE, 96 (2008), No. 5, 879–899.
[5] Mazurek P.: Optimization of Bayesian Track-Before-Detect algorithms for GPGPUs Implementations, Przegląd elektrotechniczny, 86 (2010), No. 7, 187–189.
[6] M. Borysiak, Z. Krawczyk, J. Starzyński, R. Szmurło, S. Wincenciak, CUDA accelerated finite element mesh morpher, Przegląd elektrotechniczny, 87 (2011), No. 5, 176–178.
[7] Shi H., Schmidt B., Liu W., Mueller-Wittig W.: A Parallel Algorithm for Error Correction in High-Throughput Short-Read Data on CUDA-enabled Graphics Hardware, Journal of Computational Biology, 17 (2010), No. 4, 603–615.
[8] Payne J., Sinnott-Armstrong N.A., Moore J.H.: Exploiting graphics processing units for computational biology and bioinformatics, Interdisciplinary Sciences 2 (2010), No. 3, 213–220.
[9] Liu Y., Schmidt B., Maskell D.: CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions, BMC Research Notes, 3 (2010), No. 1, 93–104.
[10] Lindholm E., Nickolls J., Oberman S., Montrym J.: NVIDIA Tesla: A Unified Graphics and Computing Architecture, IEEE Micro, 28 (2008), No. 2, 39–55.
[11] Richards F.C., Mayer T.P., Packard N.H.: Extracting Cellular automaton rules directly from experimental data, Physica D, 45 (1990), No. 1-3, 189–202.
[12] Nickolls J., Dally W.J: The GPU Computing Era, IEEE Micro, 30 (2010), No.2, 56–69.
[13] Ulery J., Computing Integer Square Roots. [web page] http://www.azillionmonkeys.com/qed/ulerysqroot.pdf. [Accessed on Aug 19, 2011].
[14] Toplak M., Curk T., Demsar J., Zupan B.: Does replication groups scoring reduce false positive rate in SNP interaction discovery?, BMC Genomics, 11(2010), No. 1, 58–62.
[15] Jakulin A., Bratko I.: Analyzing Attribute Dependencies, In: 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, 2003.

Authors: Davor Sluga (corresponding), Ph.D. Tomaž Curk, Prof. Blaž Zupan, Assoc. Prof. Uroš Lotrič, Faculty of Computer and Information Science, Univeristy of Ljubljana, Tržaška ulica 25, 1000 Ljubljana, Slovenia, e-mail: {davor.sluga, tomaz.curk, blaz.zupan, uros.lotric}@fri.uni-lj.si.