Journal of Visual Languages and Computing 42 (2017) 13–22

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

An effective and efficient approximate two-dimensional dynamic programming algorithm for supporting advanced computer vision applications

Alfredo Cuzzocrea[a,*], Enzo Mumolo[b], Giorgio Mario Grasso[c], Gianni Vercelli[d]

[a] DIA Department, University of Trieste and ICAR-CNR, Italy
[b] DIA Department, University of Trieste, Italy
[c] CSECS Department, University of Messina, Italy
[d] DIBRIS Department, University of Genova, Italy

ARTICLE INFO

Article history:
Received 7 January 2017
Accepted 11 July 2017
Available online 2 August 2017

Keywords:
Two-dimensional dynamic programming
CUDA platform
Computer vision
Intelligent systems

ABSTRACT

Dynamic programming is a popular optimization technique, developed in the 60's and still widely used today in several fields for its ability to find global optimum. *Dynamic Programming Algorithms* (DPAs) can be developed in many dimension. However, it is known that if the DPA dimension is greater or equal to two, the algorithm is an NP complete problem. In this paper we present an approximation of the fully two-dimensional DPA (2D-DPA) with polynomial complexity. Then, we describe an implementation of the algorithm on a recent parallel device based on CUDA architecture. We show that our parallel implementation presents a speed-up of about 25 with respect to a sequential implementation on an Intel I7 CPU. In particular, our system allows a speed of about ten 2D-DPA executions per second for 85×85 pixels images. Experiments and case studies support our thesis.

© 2017 Elsevier Ltd. All rights reserved.

1. Introduction

In this paper we describe an approximate *Two-Dimensional Dynamic Programming Algorithm* (2D-DPA) running on a CUDA device. *Dynamic programming* (DP), based on the Bellman's Principle of Optimality [1], is a fast, elegant method for finding the global solution to optimization problems. What characterizes a problem suitable for dynamic programming is that solutions to these problems can be formulated as a sequence of simpler problems, and the global optimum is obtained as a sequence of local optima. A classic example may be that of finding the length of a shortest path in a directed graph that has no cycles. Another classical example is that of sequence alignment. Generally-speaking, *computer vision applications* are emerging trends for such a context, and, recently, the research community has devoted a lot of attention to this topic (e.g., [2–11]).

DP has been applied to various tasks in pattern recognition and computer vision [12,13]. Nowadays, DP is considered a classic optimization method and ever though there are many other optimization techniques available, many researchers still choose DP in their optimization problems because of its conciseness, versatility, and ability to obtain globally optimal solution. Actually, DP is considered an ideal technique for solving a wide variety of discrete optimization problems such as scheduling, string editing, packaging, and inventory management. Of the recent application of DP we can mention tracking [14], stereo [15,16], and elastic image matching [17] problems. Elastic matching is a typical application of 2D-DPA.

DPA was originally developed as a continuous optimization method to obtain the solution efficiently [1]. Angel [18] used analytical DP to smooth interpolated data. Serra and Berthod [19] and Munich and Perona [20] used it for nonlinear alignment of one-dimensional patterns. Recently, Uchida et al. [21] used it in object tracking. DP matching (and its stochastic extension, i.e. Hidden Markov Models) is a classical technique for speech recognition [22] and for on-line character recognition [23].

Sequential 1D-DP matching algorithms have been extended to a two-dimensional one by many authors. Truly two-dimensional elastic image matching have been described in [24,25], but the authors have encountered the inherent NP-hardness of the problem [26]. Because of this computational intractability, practical DP-based elastic image matching algorithms employ various approximation strategies, the most popular of which is the limitation of matching flexibility, as the pseudo 2D elastic matching algorithm described in [27]. Another approximation strategy is the

* Corresponding author.
E-mail addresses: alfredo.cuzzocrea@dia.units.it (A. Cuzzocrea), mumolo@units.it (E. Mumolo), gmgrasso@unime.it (G.M. Grasso), gianni.vercelli@unige.it (G. Vercelli).

http://dx.doi.org/10.1016/j.jvlc.2017.07.002
1045-926X/© 2017 Elsevier Ltd. All rights reserved.

partial omission of the mutual dependency between 4-adjacent pixels (e.g., the tree representation in [28]). Other approximations consist in the introduction of pruning and coarse-to-fine strategies [29], at the cost of global optimality. Notwithstanding these strategies, there is currently no practical DP algorithm that can provide both globally optimal and truly two-dimensional elastic matching. All the conventional DP-based elastic matching algorithms used DP as a combinatorial optimization method. In fact a recent survey [13] reported only combinatorial (i.e., discrete) DP algorithms. Even if the DP optimization problem was originally formulated as a continuous variational problem, it has been discretized and then solved by DP as a combinatorial optimization problem [12].

The paper is organized as follows. Section 2 reports some other CUDA implementations of various DPA based applications. Section 3 describes the Dynamic Programming Algorithms, both in one and two dimensional formulations. It has been shown that the implementation of 2D-DPA has an exponential complexity, therefore in Section 4 we describe an approximation of the two-dimensional algorithm with polynomial complexity. In Section 5, we provide general architecture and functionalities of the CUDA platform. Section 6 focuses the attention on the CUDA-based implementation of approximate DPA. Section 7 reports experiments showing the benefits that derive from our proposed algorithm. In Section 8, we report a complete case study and related experimental results obtained from the application of the algorithm to fingerprint verification. Finally, in Section 9, we report concluding remarks and future work.

2. Related work

In this Section, we provide an overview of state-of-the-art proposals related to our research. Since the sequential implementation of various types of DPA has high computational demand, many authors implemented the algorithm on Graphics Processing Devices. Two issues have been mainly considered: how to find the best way to parallelize the DPA itself and how to parallelize the problem which has to be solved with DPA.

Many problems have been solved with DPA. The most popular are Stereo Matching in stereo vision, Elastic Matching of images, or various discrete numerical calculus problems. In 2007, a Dynamic Programming-based low density real-time Stereo Matching was implemented on an ATI Radeon $X800$, an early GPU device. They obtained a frame rate from 10 to 20 fps [30].

In 2009, Xiao et al. address the problem of mapping DPA on *Graphics Processing Units*. They propose a fine-grained parallelization of a single instance of the algorithm that is mapped to the GPU. Steffen et al. [31] describe in 2010 an implementation, on a GTX 280, of a numerical framework, called *Algebraic Dynamic Programming*, for encoding a broad range of optimization problems. Depending on the application, they report speed ups ranging from about 6 to about 25. In the same year, Congote et al. [32] describe the implementation of a *Dense Stereo Matching* algorithm based on Dynamic Programming to recover depth map from two-dimensional images using dynamic programming. They used a number of GPU's available in that year for a parallel implementation of the dynamic programming based algorithm. The sequential implementation was performed with an Intel Pentium processor $E2180$. They found a speed-up of about 16 between the two devices. Stivala et al. [33] published in 2010 a paper showing how to parallelize any DPA on a shared memory multi-core computer by means of a shared lock-free hash table, via starting multiple threads that compute the DP recursion in a top-down fashion and memorizing the result in a shared lock-free hash table.

In 2011, Wu et al. [34] present the GPU acceleration of an important category of DP problems, called *Non-Serial Polyadic Dynamic Programming*. Since in these problems the parallelism level varies significantly in different stages of computation, they adjusted the thread-level parallelism in mapping a NPDP problem onto the GPU. They report a speed up of about 13 over the previously published GPU algorithm. Nishida et al. in 2012 solved an optimization problem with a known dynamic programming solution on a NVIDIA GeForce GTX 580. The problem was the computation of the optimal polygon triangulation of a convex polygon with minimum total weight. The algorithm they published in [35] attained a very high speed up factor of about 250.

3. One- and two-dimensional DPA

In this Section, we focus the attention on one- and two-dimensional DPA. A popular way to describe *One-Dimensional Dynamic Programming Algorithms* (1D-DPA) is by means of the *Edit Distance* [36]. The Edit Distance, which finds applications in bio-informatics [37], natural language processing [38] and spoken-word recognition [22], is a way to measure the similarity of two strings or, in other words, to align the two strings. In the following description we extend the Edit algorithm to the comparison of one-dimensional sequences, similar to the comparison between spoken words [22].

Given two one-dimensional sequences, $A = (a_1, a_2, \ldots, a_i, \ldots, a_N)$ and $B = (b_1, b_2, \ldots, \ldots, b_j, \ldots b_M)$, the mapping of one sequence to the other is represented by a path M' which starts from cell (1, 1) to cell (N, M). The path is formed by a number of points so that each point k of the path corresponds to a couple of coordinates, $M_k = (i_k, j_k)$. A distance between the two sequences can be defined by the sum of the local distances between the elements of the sequences, a_i, b_j, computed along a path, namely: $\sum_{k=1}^{|M'|} \left\| a_{i_k} - b_{j_k} \right\|$, where $|M'|$ is the length of the path M'. Clearly, there exists a path along which the cumulative distance is minimum. In this case the cumulative distance is the distance between the two sequences:

$$D(A, B) = \frac{\min_{M'} \sum_{k=1}^{|M'|} d(M'_k)}{|M'|} = \frac{\min_{M'} \sum_{k=1}^{|M'|} d(i_k, j_k)}{|M'|} = \frac{\min_{M'} \sum_{k=1}^{|M'|} \left\| a_{i_k} - b_{j_k} \right\|}{|M'|} \quad (1)$$

It is worth noting that the factor at the denominator is needed to normalize the distance against different lengths of the optimum path, and it is needed when Eq. (1) is used to measure the distance between images.

By Dynamic Programming, the optimization problem of (1) is solved by updating the cumulative distance $D(i, j)$ at each point of the $A - B$ space using the recursion described in Eq. (2), which performs the optimal principle of DP.

$$D(i, j) = min \begin{cases} D(i-1, j) + d(i, j) \\ D(i-1, j-1) + 2d(i, j) \\ D(i, j-1) + d(i, j) \end{cases} \quad (2)$$

where $D(1, 1) = 2d(1, 1)$. The DP recursion described in Eq. (2) is represented by in Fig. 1.

After examination of the $A - B$ space, Eq. (1) becomes Eq. (3):

$$D(A, B) = \frac{D(N, M)}{N + M} \quad (3)$$

where $D(N, M)$ is the cumulative distance at the point (N, M), and the fact that the length of the optimum path is $N + M$ is due to the weight of 2 on the diagonal move. It is important to note that M' which corresponds to the minimum cumulative distance $D(N, M)$ is the optimal map between A and B and can be used to align one sequence on the other. This operation is called warping. We can

Download English Version:

https://daneshyari.com/en/article/4968148

Download Persian Version:

https://daneshyari.com/article/4968148

Daneshyari.com