Information Systems ■ (■■■■) ■■■–■■■

Contents lists available at ScienceDirect

Information Systems

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

Improving matrix-based dynamic programming on massively parallel accelerators ☆

David Bednárek, Michal Brabec, Martin Kruliš *

Parallel Architectures/Algorithms/Applications Research Group, Faculty of Mathematics and Physics, Charles University in Prague, Malostranské nám. 25, Prague, Czech Republic

ARTICLE INFO

Article history:
Received 1 December 2015
Received in revised form
6 May 2016
Accepted 3 June 2016

Keywords:
Parallel
Multicore
GPU
Intel Xeon Phi
Dynamic programming
Edit distance
Dynamic time warping

ABSTRACT

Dynamic programming techniques are well-established and employed by various practical algorithms, including the edit-distance algorithm or the dynamic time warping algorithm. These algorithms usually operate in an iteration-based manner where new values are computed from values of the previous iteration. The data dependencies enforce synchronization which limits possibilities for internal parallel processing. In this paper, we investigate parallel approaches to processing matrix-based dynamic programming algorithms on modern multicore CPUs, Intel Xeon Phi accelerators, and general purpose GPUs. We address both the problem of computing a single distance on large inputs and the problem of computing a number of distances of smaller inputs simultaneously (e.g., when a similarity query is being resolved). Our proposed solutions yielded significant improvements in performance and achieved speedup of two orders of magnitude when compared to the serial baseline.

1. Introduction

Dynamic programming is a well established technique employed when a problem is defined using a recursive formula whose naïve application would lead to an algorithm with exponential time complexity. If the subproblems overlap, the dynamic programming approach can prune out redundant work, so that each subproblem is computed at most once and the time complexity is reduced to polynomial. On the other hand, algorithms based on dynamic programming are usually difficult to parallelize, since the subproblems are interdependent – i.e., each subproblem requires the results of previous subproblems.

In this work, we focus on dynamic programming algorithms whose subproblems can be organized in a two-dimensional matrix. Each partial result in the matrix is computed from a small subset of previous results, which permits a limited degree of concurrent evaluation. For instance, if $x_{i,j}$ is the result value of subproblem (i,j) and $f_{i,j}$ is the function computing it, the problem formula may look like

$$x_{i,j} = f_{i,j}(x_{i-1,j}, x_{i-1,j-1}, x_{i,j-1}).$$

Typical examples of such algorithms are the Wagner–Fischer dynamic programming algorithm [3] for the edit distance problem originally described by Levenshtein [4], the dynamic time warping [5] (DTW), or the Smith–Waterman algorithm [6] for molecular sequence alignment. The matrix shape of their subproblem spaces stems from the fact that these algorithms are designed to

☆ This paper is an extension of two previous papers *On Parallel Evaluation of Matrix-Based Dynamic Programming Algorithms* [1] and *Improving Parallel Processing of Matrix-based Similarity Measures on Modern GPUs* [2] of the same authors. In this extension, we describe the problem and our solution in more detail, provide an analysis of the data dependencies, include multi-distance solution for multicore CPUs and Xeon Phi devices, and present significantly more thorough empirical evaluation and comparison of utilized platforms.

* Corresponding author.
E-mail addresses: bednarek@ksi.mff.cuni.cz (D. Bednárek), brabec@ksi.mff.cuni.cz (M. Brabec), krulis@ksi.mff.cuni.cz (M. Kruliš).

http://dx.doi.org/10.1016/j.is.2016.06.001
0306-4379/

compare a pair of sequences; this purpose in turn implies their significance as a distance or similarity measure in similarity search.

In parallel programming, the applicability of a particular parallelizing strategy depends on many factors, including typical data sizes, usage scenarios and, of course, the hardware architecture. In this paper, we study two scenarios which occur in similarity search: evaluating distance of a pair of sequences and simultaneous evaluation of a set of distances between a given (*query*) sequence and a set of (*database*) sequences. We investigated the two scenarios in three parallel hardware environments: Multicore CPUs with SIMD[1] support, manycore CPUs (Intel Xeon Phi), and GPU devices.

1.1. Problem details

We have selected the Wagner–Fischer dynamic programming algorithm [3] for the Levenshtein distance problem as a representative for our implementation since its computational simplicity emphasizes the communication and synchronization overhead associated with parallel computations. However, we will not employ any optimizations designed specifically for the Levenshtein distance (like the Myers' algorithm [7]), so our proposed improvements are applicable for other dynamic programming algorithms with similar data-dependency pattern.

Functions $f_{i,j}$ employed in these distances are often very simple. In the case of the Wagner–Fischer dynamic programming algorithm [3] for the Levenshtein distance, the function involves only comparison, incrementation, and minimum:

$$f_{i,j}(p,q,r) = \min(p+1, q+1-\delta_{u[i]}^{v[j]}, r+1),$$

where the Kronecker δ compares the i-th and j-th positions in the input strings u and v respectively.

The dependencies between individual invocations of the formula f significantly limit the parallelism available in the problem. For a $M \times N$ matrix (i.e., inputs of size M and N respectively), at most $\min(M,N)$ elements may be computed in parallel using the diagonal approach illustrated in Fig. 1. Therefore, when the computation of f does not take orders of magnitude more time than the data exchange, a pure diagonal approach cannot be effectively employed on current CPU and GPU architectures. On the other hand, the diagonal approach gives us basis for processing subsections of the distance matrix using SIMD instructions (in the case of CPU and Intel Xeon Phi) or threads running in lockstep (in the case of GPU).

Current parallel architectures employ multiple levels of parallelism. The CPU architectures offer multiple cores processing independent threads, while each core also offers special SIMD instructions for low-level data parallelism. Similarly, the GPUs are composed of multiprocessors, which can process different kernels, while each multiprocessor is comprised of tightly coupled cores, which share most of the resources including instruction schedulers. Therefore, our solution will also present a model which employs two levels of parallelism.

Finally, let us point out that in most applications including similarity measures, the only expected result is the computed distance, which is the bottom-right element of the matrix. Hence, the matrix does not have to be completely materialized in memory. For instance, if problem is being solved by the diagonal method presented in Fig. 1, only values of two last diagonals are required to compute next diagonal. This observation is quite important for our implementation as it allows us to perform optimizations such as keeping matrix values only in registers or specialized caches (like shared memory of the GPU).

1.2. Contributions and paper structure

This paper investigates the problems of matrix-based dynamic programming similarity measures on current parallel architectures, namely

- multicore CPUs,
- Intel Xeon Phi parallel accelerators,
- and general purpose GPUs.

The basic idea is based on aggregating subtasks of the matrix into regular parallelograms, which can be processed independently and also parallelized internally. A GPU implementation using this technique was first presented by Tomiyama et al. [8]. We have improved the original approach in several ways:

- We have tested the algorithm on current hardware and adjusted its parameters for current generations of GPUs.
- New optimizations were employed including better data caching and more efficient data exchange using new warp shuffle instructions (introduced in CUDA compute capability 3.0).
- The approach was adopted for multicore CPUs and Intel Xeon Phi devices, where GPU thread warps (i.e., threads running in lockstep) are replaced with SIMD instructions.

In addition, we have investigated possible parallelization approaches to a multi-distance problem – i.e., when there are multiple *jobs* to be computed in parallel where each job consists of computing the distance of a pair of objects. This situation is typical for similarity queries, when distances between a query and all database objects need to be computed, or when pivot-based index is being constructed and distances between vantage objects and database objects are being computed. Even though this may seem to be an embarrassingly data parallel problem, the associated ratio between the memory and the CPU loads imposes serious performance limitations when executed on manycore GPUs and Intel Xeon Phi devices. Thus, we investigated the necessary trade-off between the memory-consuming parallelism among independent distance-computing jobs and the limited parallelism available inside one job.

[1] Single instruction multiple data parallel paradigm.

Please cite this article as: D. Bednárek, et al., Improving matrix-based dynamic programming on massively parallel accelerators, Information Systems (2016), http://dx.doi.org/10.1016/j.is.2016.06.001