Theoretical Computer Science 700 (2017) 1–22

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Parallel construction of succinct trees [☆], [☆☆]

José Fuentes-Sepúlveda [a,*], Leo Ferres [b], Meng He [c], Norbert Zeh [c]

[a] Department of Computer Science, Universidad de Chile, Santiago, Chile
[b] Faculty of Engineering, Universidad del Desarrollo & Telefónica I+D, Santiago, Chile
[c] Faculty of Computer Science, Dalhousie University, Halifax, Canada

ARTICLE INFO

Article history:
Received 2 September 2016
Received in revised form 9 June 2017
Accepted 29 July 2017
Available online 3 August 2017
Communicated by C. Kaklamanis

Keywords:
Succinct data structure
Succinct tree construction
Multicore
Parallel algorithm

ABSTRACT

Succinct representations of trees are an elegant solution to make large trees fit in main memory while still supporting navigational operations in constant time. However, their construction time remains a bottleneck. We introduce two parallel algorithms that improve the state of the art in succinct tree construction. Our results are presented in terms of *work*, the time needed to execute a parallel computation using one thread, and *span*, the minimum amount of time needed to execute a parallel computation, for any amount of threads. Given a tree on n nodes stored as a sequence of balanced parentheses, our first algorithm builds a succinct tree representation with $O(n)$ work, $O(\lg n)$ span and supports a rich set of operations in $O(\lg n)$ time. Our second algorithm improves the query support. It constructs a succinct representation that supports queries in $O(c)$ time, taking $O(n + \frac{n}{\lg^c n}\lg(\frac{n}{\lg^c n}) + c^c)$ work and $O(c + \lg(\frac{nc^c}{\lg^c n}))$ span, for any positive constant c. Both algorithms use $O(n\lg n)$ bits of working space. In experiments using up to 64 cores on inputs of different sizes, our first algorithm achieved good parallel speed-up. We also present an algorithm that takes $O(n)$ work and $O(\lg n)$ span to construct the balanced parenthesis sequence of the input tree required by our succinct tree construction algorithm.

1. Introduction

Trees are ubiquitous in Computer Science. They have applications in every aspect of computing from XML/HTML processing to abstract syntax trees (AST) in compilers, phylogenetic trees in computational genomics or shortest path trees in path planning. The ever increasing amounts of structured, hierarchical data processed in many applications have turned the processing of the corresponding large tree structures into a bottleneck, particularly when they do not fit in memory. Succinct tree representations store trees using as few bits as possible and thereby significantly increase the size of trees that fit in memory while still supporting important primitive operations in constant time. There exist such representations that use only $2n + o(n)$ bits to store the topology of a tree with n nodes [2–8], which is close to the information-theoretic lower bound and much less than the space used by traditional pointer-based representations.

[☆] A previous version of this paper appeared in the 14th International Symposium on Experimental Algorithms (SEA 2015) [1].

[☆☆] This work was supported by the Emerging Leaders in the Americas scholarship programme, NSERC, and the Canada Research Chairs programme. The first author received funding from FONDECYT 3170534 and Basal Funds FB0001, CONICYT. The second author would like to thank Telefónica I+D Chile and the Chilean government initiative CORFO 13CEE2-21592 (2013-21592-1-INNOVA_PRODUCCION2013-21592-1) for financial support.

[*] Corresponding author.
E-mail addresses: jfuentess@dcc.uchile.cl (J. Fuentes-Sepúlveda), lferres@udd.cl (L. Ferres), mhe@cs.dal.ca (M. He), nzeh@cs.dal.ca (N. Zeh).

http://dx.doi.org/10.1016/j.tcs.2017.07.025
0304-3975/© 2017 Elsevier B.V. All rights reserved.

Alas, the construction of succinct trees is quite slow compared to the construction of pointer-based representations. Multicore parallelism offers one possible tool to speed up the construction of succinct trees, but little work has been done in this direction so far. The only results we are aware of focus on the construction of wavelet trees, which are used in representations of text indexes. In [9], two practical multicore algorithms for wavelet tree construction were introduced. Both algorithms perform $O(n \lg \sigma)$[1] work and have $O(\lg n)$ span, where n is the input size, σ is the alphabet size, work is the time needed to execute a parallel computation using a single thread, and span is the minimum time needed to execute a parallel computation for any number of threads. In [10–12], Shun introduced new algorithms to construct wavelet trees in parallel. Among these algorithms, the best algorithm in practice performs $O(n \lg \sigma)$ work and has $O(\lg n \lg \sigma)$ span. Shun also explained how to parallelize the construction of rank/select structures so that it requires $O(n)$ work and $O(\lg n)$ span for rank structures, and $O(n)$ work and $O(\lg n)$ span for select structures.

In this paper, we provide a parallel algorithm that constructs the RMMT tree representation of [2] in $O(n)$ work and $O(\lg n)$ span. This structure is a simplified version of the succinct tree representation in [2], and it uses $2n + o(n)$ bits to store an ordinal tree on n nodes and supports a rich set of basic operations on these trees in $O(\lg n)$ time. While this query time is theoretically suboptimal, the RMMT structure is simple enough to be practical and has been verified experimentally to be very small and support fast queries in practice [13]. Combined with the fast parallel construction algorithm presented in this paper, it provides an excellent tool for manipulating very large trees in many applications.

We implemented and tested our algorithm on a number of real-world input trees having billions of nodes. Our experiments show that our algorithm run on a single core is competitive with state-of-the-art sequential constructions of the RMMT structure and achieves good speed-up on up to 64 cores and likely beyond.

We then designed a parallel algorithm to construct the more complex, optimal succinct tree representation that supports operations in $O(c)$ time using $2n + O(n/\lg^c n)$ bits, for any constant $c > 0$. This algorithm has $O(n + \frac{n}{\lg^c n} \lg(\frac{n}{\lg^c n}) + c^c)$ work and $O(c + \lg(\frac{nc^c}{\lg^c n}))$ span. In the design of this new algorithm, we provide parallel algorithms to construct the 2d-Min-Heap data structure and the ladder decomposition of trees. We think that those partial results may be of independent interest. For example, the ladder decomposition is used to support level ancestor in $O(1)$ time [14].

The remainder of this paper is organized as follows: Section 2 gives a brief overview of the RMMT structure, to clearly define its structure and illustrate how it can be used to support basic operations on trees efficiently. It also briefly discusses other previous work on succinct tree representations and reviews the dynamic multithreading model, which we use to analyze the theoretical running time of our algorithm. Section 3 describes our parallel algorithms for constructing succinct representations of trees and for computing the balanced parentheses representation of trees. Section 4 discusses our experimental setup and results. Section 5 offers concluding remarks and discusses future work.

2. Preliminaries

2.1. Succinct trees

Jacobson [3] was the first to propose the design of succinct data structures. He showed how to represent an ordinal tree on n nodes using $2n + o(n)$ bits so that computing the first child, next sibling or parent of any node takes $O(\lg n)$ time in the bit probe model. Clark and Munro [4] showed how to support the same operations in constant time in the word RAM model with word size $\Theta(\lg n)$. Since then, much work has been done on succinct tree representations, to support more operations, to achieve compression, to provide support for updates, and so on [15–19,5,6,2]. See [7] for a thorough survey.

Navarro and Sadakane [2] proposed a succinct tree representation, referred to as NS-representation throughout this paper, which was the first to achieve a redundancy of $O(n/\lg^c n)$ bits for any positive constant c. The *redundancy* of a data structure is the additional space it uses above the information-theoretic lower bound. While all previous tree representations achieved a redundancy of $o(n)$ bits, their redundancy was $\Omega(n \lg \lg n / \lg n)$ bits, that is, just slightly sub-linear. The NS-representation also supports a large number of navigational operations in constant time (see Table 1); only the work in [5,6] supports two additional operations: level_leftmost, which finds the leftmost node at a given level, and level_successor(x), which finds the node immediately to the right of node x at the same level. An experimental study of succinct trees [13] showed that a simplified version of the NS-representation uses less space than other existing representations in most cases and performs most operations faster. In this paper, we provide a parallel algorithm for constructing this representation.

2.1.1. Simplified NS-representation

The NS-representation is based on the balanced parenthesis sequence P of the input tree T, which is obtained by performing a preorder traversal of T and writing down an opening parenthesis when visiting a node for the first time and a closing parenthesis after visiting all its descendants. Thus, the length of P is $2n$. See Fig. 1 as an example.

The NS-representation is not the first structure to use balanced parentheses to represent trees. Munro and Raman [15] used succinct representations of balanced parentheses to represent ordinal trees and reduced a set of navigational operations on trees to operations on their balanced parenthesis sequences. Their solution supports only a subset of the operations

[1] We use $\lg x$ to mean $\log_2 x$ throughout this paper.