MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

A Parallel Tree Contraction Algorithm on Non-Binary Trees

Akimasa MORIHATA and Kiminori MATSUZAKI

METR 2008–27 June 2008

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

WWW page: http://www.keisu.t.u-tokyo.ac.jp/research/techrep/index.html

A Parallel Tree Contraction Algorithm on Non-Binary Trees

Akimasa MORIHATA and Kiminori MATSUZAKI

Abstract

Parallel tree contraction is an important framework to develop efficient parallel algorithms on trees. Parallel tree contraction gives an appropriate scheduling for parallel computations on trees, and the scheduling brings efficient parallel algorithms to us. While there are many studies for efficient algorithms of parallel tree contraction and implementation of various parallel computations based on them, few studies give practical parallel tree contraction algorithms on non-binary trees. Some studies just mention that any tree can be encoded as a binary tree; however, binary-tree encoding breaks the original structure, and this makes it difficult to develop parallel algorithms following parallel tree contraction algorithms.

In this paper, we propose a new parallel tree contraction algorithm. Our algorithm is a generalization of that proposed by Abrahamson et al., and works well even for non-binary trees. Our algorithm requires no binary-tree encoding, and thus, it is easy to develop parallel algorithms based on our parallel tree contraction algorithm. In addition, we show sufficient conditions when computations can be parallelized based on our algorithm, and the conditions are generalizations of those known on binary trees.

1 Introduction

The objective of this paper is to give a way to develop efficient parallel algorithms on trees. One naive way to parallelize computations on trees is to evaluate independent subtrees in parallel. However, such naive parallelization results in miserable efficiency if the tree forms an ill-balanced shape such as a monadic one. It is difficult to give parallel algorithms that are efficient even for ill-balanced trees.

Parallel tree contraction, introduced by Miller and Reif [MR85], is a framework of constructing efficient parallel algorithms on trees. The parallel tree contraction problem is to give a scheduling of contraction operations so that they can collapse a tree efficiently in parallel with no conflict. Once an efficient parallel tree contraction algorithm is given, we can achieve many computations on a tree efficiently in parallel by processing computations according to the scheduling. Firstly Miller and Reif introduced the notion of parallel tree contraction to obtain an efficient parallel algorithm for evaluating expressions defined with $+$, $-$, $\times$, and $/$. After that, parallel tree contraction is recognized as an important framework for constructing various parallel algorithms on trees. Many studies have been done for efficient parallel tree contraction algorithms [CV88, GR89, ADKP89, MW97] and for implementation of many computations based on them [DK92, GCS94, Ski96, MHT06, Mat07].

Although the importance of parallel tree contraction is well recognized, few studies consider that on non-binary trees. Some studies just mentioned that non-binary trees can be encoded as binary trees. However, such encoding is often troublesome because it breaks the original structure. For example, consider that we want to compute the height of a non-binary tree, and we encode the tree as a binary tree. Then, the height of the binary tree is not that of the original tree anymore. Moreover, we have several binary-tree encodings for a single tree, and we need to select an appropriate binary-tree encoding to develop an efficient parallel algorithm on it. In short, the binary-tree encoding makes problems complicated.

In this paper, we give a new parallel tree contraction algorithm that works well even for non-binary trees. Our algorithm is a generalization of that proposed by Abrahamson et al. [ADKP89]. One important characteristic of our algorithm is that it requires no binary-tree encoding. For this characteristic, it is easy to develop parallel algorithms based on our parallel tree contraction algorithm. Our algorithm runs in $O(kn/p + k \log p)$ time on EREW PRAM machines, where n is the size of the tree, p is the number of processors, and k is the maximum degree of the tree. In addition, we show sufficient conditions when computations can be parallelized based on our algorithm. Our sufficient conditions are generalizations of those known on binary trees [GR89, ADKP89].

2 Related Works

First, Miller and Reif [MR85] introduced the notion of parallel tree contraction and showed a tree contraction algorithm. Although the algorithm also works for non-binary trees, it assumes CRCW PRAM for the parallel computation model. In [Rei93], a cost-optimal tree contraction algorithm on EREW PRAM machines is shown, which is an extension of that by Miller and Reif. Our algorithm is simpler than that algorithm and suitable for the practical use.

While there are several efficient parallel tree contraction algorithms on binary trees [CV88, GR89,ADKP89,MW97], few studies consider parallel tree contraction algorithms on non-binary trees without binary-tree encoding. In fact, binary-tree encoding is not troublesome in parallel tree contraction itself; however, such encoding breaks the structure of the original trees and makes it hard to develop parallel algorithms based on parallel tree contraction.

As an application on non-binary trees, some studies apply parallel tree contraction to parallel term matching algorithms, for example [DK92, MW97]. Most studies developed algorithms on binary-tree encodings even in the case that terms form non-binary trees. It is not clear whether parallel term matching algorithms can be generalized so that it can deal with other computations on non-binary trees.

Skillicorn [Ski96] proposed to use parallel tree contraction for implementing parallel tree skeletons. Parallel tree skeletons are general and expressive patterns in parallel computation, and many computations on trees can be developed with them [GCS94,MHT06,Mat07]. However, most studies only deal with binary trees. Matsuzaki et al. [MHKT05, KME07, Mat07] proposed parallel tree skeletons on non-binary trees, which are implemented based on a binary-tree encoding. They showed a sufficient condition, called extended distributivity, as a requirement of a successful parallel implementation on the binary-tree encoding. Our sufficient conditions are simpler and more understandable than their condition.

3 SHUNT Contraction Algorithm

We consider the EREW PRAM model. An input tree is given by a set of nodes, and each node has pointers to its parent and children.

Here we introduce a parallel tree contraction algorithm by Abrahamson et al. [ADKP89], called the SHUNT[1] contraction algorithm. The problem is to collapse a tree in $O(\log n)$ steps, where a step consists of a set of independent SHUNT operations defined below.

Definition 1 (SHUNT)**.** Specified a leaf, a SHUNT operation removes the leaf and its parent and connects the sibling of the leaf to its grandparent. □

[1]The name "SHUNT" is later given in [Rei93].

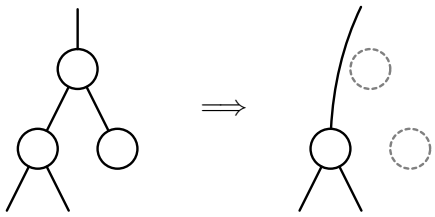

Figure 1: A SHUNT operation

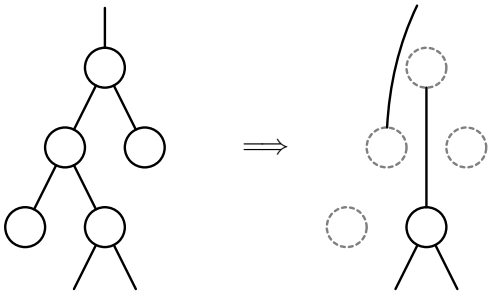

Figure 2: A conflict of two SHUNT operations

Figure 1 shows the behavior of a SHUNT operation. We call two SHUNT operations are independent if no nodes concern them simultaneously. On one hand, we require the SHUNT operations to be independent on the EREW PRAM; otherwise, the tree structure will be broken as shown in Figure 2. On the other hand, we should apply many SHUNT operations simultaneously to accomplish the reduction in $O(\log n)$ steps. Therefore, we need to give a good conflict-free scheduling of SHUNT operations. Abrahamson et al. [ADKP89] showed that a numbering on leaves resolves this problem. The following procedure gives a scheduling of independent SHUNT operations and finishes the reduction in $O(\log n)$ steps.

Procedure 2 (SHUNT contraction for binary trees)**.**

(1) Number all leaves from left to right.
(2) Do SHUNT for all odd-numbered left leaves.
(3) Do SHUNT for all odd-numbered right leaves.
(4) Halve all numbers of leaves.
(5) Go to (2) until the tree consists of only one node. □

Next let us consider computations on binary trees. When the whole computation can be achieved by a sequence of small-step computations corresponding to SHUNT operations, we can do the computation efficiently in parallel according to the scheduling given by Procedure 2. Here we introduce *algebraic computations* as a general computation pattern on trees. Note that the following algebraic computations are defined on not only binary trees but also non-binary trees whose degree is fixed.

Definition 3 (algebraic computations [ADKP89])**.** A set of values S and a set of functions F are given, where each element of F takes a fixed-sized tuple of elements of S according to its arity and results in an element of S. An *algebraic computation* defined by (S, F) is to evaluate expressions whose operators are elements of F and values are elements of S. □

Abrahamson et al. gave a sufficient condition for parallelizing algebraic computations where arities of operators are two.

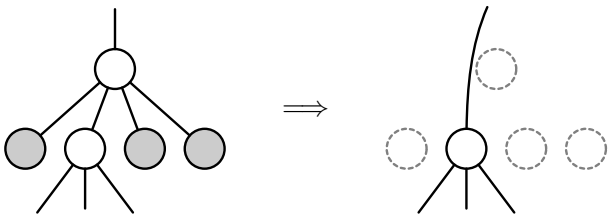

Figure 3: An M-SHUNT operation (marked nodes are colored)

Theorem 4. Assume that there are a set of values S and two sets of indexed functions $F \subseteq (S \times S \to S)$ and $G \subseteq (S \to S)$ such that the following conditions hold.

- Any element of F and G can be evaluated in O(1) time.
- For all $f_i \in F$ and $a, b \in S$, there exist functions g_j and g_k such that $f_i(x, b) = g_j(x)$ and $f_i(a, x) = g_k(x)$ hold and the indexes j and k can be computed in O(1) time from i, a, and b.
- For all $g_i, g_j \in G$, there exists a function g_k such that $g_i(g_j(x)) = g_k(x)$ holds and the index k can be computed in O(1) time from i and j.

Then, an algebraic computation defined by (S, F) can be computed in $O(n/p + \log p)$ time in parallel, where n is the size of the expression and p is the number of processors. □

4 Parallel Tree Contraction Algorithm for Non-Binary Trees

Now let us consider non-binary cases. We assume that a tree has no unary node; otherwise, we can remove such a node by inserting a dummy leaf.

In the case of binary trees, a leaf has a unique sibling and thus we can define the SHUNT operation to be applied to a leaf. In the case of non-binary trees, however, a single leaf does not necessarily specify its sibling nor a SHUNT operation. Here, we introduce *marks* on leaves to generalize the SHUNT operation for non-binary trees.

Definition 5 (SHUNT operation with marks)**.** A SHUNT operation with marks (we will call it *M-SHUNT*) is an operation applied to an internal node whose children are one unmarked node and the other marked leaves. An M-SHUNT operation removes the internal node and all its marked children and connects the unmarked child to the parent of the internal node. □

Figure 3 shows the behavior of an M-SHUNT operation. Note that an M-SHUNT operation to a binary internal node is isomorphic to the usual SHUNT operation. We assume that an M-SHUNT operation for a k-ary node takes $O(k)$ time.

The following procedure is our parallel tree contraction algorithm for non-binary trees.

Procedure 6 (SHUNT contraction for non-binary trees)**.**

(1) Number all leaves from left to right.
(2) Mark all odd-numbered leaves that have an unmarked right sibling, and apply M-SHUNT operations to all the possible nodes.
(3) Mark all odd-numbered unmarked leaves, and apply M-SHUNT operations to all the possible nodes.
(4) Erase the numbers of the marked leaves, and halve those of the unmarked leaves.
(5) Go to (2) until the tree consists of only one node. □

It is worth noting that Procedure 6 is equivalent to Procedure 2 when the input is a binary tree. In this sense, we can state that Procedure 6 is a generalization of Procedure 2. Actually, Procedure 6 inherits good characteristics from Procedure 2.

Lemma 7. Procedure 6 raises no conflicting applications of M-SHUNT operations.

Proof. Note that unmarked leaves are numbered from left to right throughout the procedure.

Let v_1 be the parent of an internal node v_2.

First we prove by contradiction that simultaneous M-SHUNT operations to v_1 and v_2 never occur in the step (2). Let l_1 be the rightmost newly marked leaf of v_1 and l_2 be the leftmost newly marked leaf of v_2. By the assumption that M-SHUNT operations are applicable to both v_1 and v_2, v_1 has exactly one unmarked child that is v_2, and v_2 has the only unmarked child on the right of l_2. Because l_1 is newly marked and v_2 is the only unmarked child of v_1, v_2 is a right sibling of l_1, and thus the number of l_1 is less than that of l_2. Since even-numbered leaves remain unmarked, an unmarked even-numbered leaf (say l_3) should exist between l_1 and l_2. Here, l_3 is not a child of v_1 because v_1 has the only unmarked child v_2; l_3 is not a descendant of v_2 because v_2 should have the unmarked child on the right of l_2 but not on the left of l_2. Therefore, such a leaf l_3 must not exist and a contradiction occurs.

The case for the step (3) is similar. Let l_1 be the leftmost newly marked leaf of v_1, l_2 be the rightmost newly marked leaf of v_2, and l_3 be an even-numbered leaf between l_1 and l_2. Notice that since v_2 is an unmarked child of v_1 and l_1 is not marked in the previous step, l_1 is a right sibling of v_2 and the number of l_1 is greater than that of l_2. Obviously l_3 is not a child of v_1. If l_3 is a child of v_2, then l_3 is an unmarked right sibling of l_2 due to the order of l_1 and l_2, but in such a case l_2 should be marked in the previous step (2), which is a contradiction. □

Theorem 8. Procedure 6 runs in $\mathrm{O}(kn/p + k \log p)$ time on EREW PRAM machines, where n is the size of the tree, p is the number of processors, and k is the maximum degree of the tree.

Proof. Correctness on the EREW PRAM follows from Lemma 7.

The step (1) can be implemented by the Euler-tour technique with the list ranking procedure and is done in $\mathrm{O}(kn/p + k \log p)$ time. Since we assumed the cost of an M-SHUNT operation to be $\mathrm{O}(k)$, the cost of the theorem is achieved if the steps (2)–(5) take $\mathrm{O}(n/p + \log p)$ steps. Since the number of unmarked leaves decreases into the half through a sequence of the steps (2)–(5), the cost is achieved as the case of Procedure 2. □

Notice that the complexity is cost optimal when p is $\mathrm{O}(n/\log n)$ and k is $\mathrm{O}(1)$, that is, $\mathrm{O}(p(kn/p + k \log p)) = \mathrm{O}(n)$ holds.

One important fact is that Procedure 6 requires no binary-tree encoding; thus, it is easy to develop parallel algorithms based on the algorithm. The following theorem shows a sufficient condition to achieve parallel computations following the Procedure 6, which is a generalization of Theorem 4.

Theorem 9. Assume that there are a set of values S and two sets of indexed functions F and G such that the following conditions hold.

- Any element of F and G can be evaluated in $\mathrm{O}(1)$ time.
- For all $f_i \in F$ and $a_1, a_2, \ldots, a_{l-1}, a_{l+1}, \ldots, a_k \in S$, where k is the arity of f_i, there exists a function g_j such that $f_i(a_1, \ldots, a_{l-1}, x, a_{l+1}, \ldots, a_k) = g_j(x)$ holds and the index j can be computed in $\mathrm{O}(k)$ time from $a_1, \ldots, a_{l-1}, a_{l+1}, \ldots, a_k$, l, and i.
- For all $g_i, g_j \in G$, there exists a function g_k such that $g_i(g_j(x)) = g_k(x)$ holds and the index k can be computed in $\mathrm{O}(1)$ time from i and j.

Then, an algebraic computation defined by (S, F) can be computed in $O(n/p + \log p)$ time in parallel, where n is the size of the expression and p is the number of processors.

Proof. We can assume that G contains the identity function without loss of generality. As a preprocess, associate the index of the identity function to each internal node. After that, run the Procedure 6, and when M-SHUNT operation is applied to a node, do computation described bellow. Let $f_j \in F$ and p respectively be the operator and the index stored at the node. If all children of the node are leaves whose values are $a_1, \ldots, a_k$, store $g_p(f_j(a_1, \ldots, a_k))$ to the leaf left after the M-SHUNT operation. If l-th child of the node is an internal node that stores an index q and values of other children are $a_1, \ldots, a_{l-1}, a_{l+1}, \ldots a_k$, update the index q by r such that $g_r(x) = g_p(f_j(a_1, \ldots, a_{l-1}, g_q(x), a_{l+1}, \ldots, a_k))$ holds.

It is not difficult to see that the procedure above yields the result of the algebraic computation; besides, because arity of each function in F is at most constant, it runs in $O(n/p + \log p)$ time. □

A direct consequence of Theorem 9 is a parallel evaluation algorithm for algebraic computations whose carrier is finite.

Corollary 10. If the size of the set S is $O(1)$ and each element of F can be evaluated in $O(1)$ time, any algebraic computation defined by (S, F) can be computed in $O(n/p + \log p)$ time in parallel, where n is the size of the expression and p is the number of processors.

Proof. Let each functions in G required in Theorem 9 be a transition table from S to S. Since the size of S is $O(1)$, the size of a table is $O(1)$ and a composition of two tables can be evaluated in $O(1)$ time. □

Corollary 10 is a generalization of the known result where operators are binary [GR89]. Corollary 10 gives, for example, an efficient parallel evaluation algorithm for arithmetic expressions consisting of $+$, $-$, $\times$, $/$, and conditional operators, on a Galois field.

5 Conclusion

In this paper, we described a generalization of the known results about the parallel tree contraction and its condition for applications on binary trees [GR89, ADKP89] so that it can efficiently cope with non-binary trees of bounded degrees. We gave a cost optimal parallel tree contraction algorithm for non-binary trees, and gave a sufficient condition for parallelizing algebraic computations that form non-binary trees.

Our algorithm is not efficient when some internal nodes have many children. It is a topic of further research to give an efficient algorithm that works well for such trees.

References

[ADKP89] Karl R. Abrahamson, N. Dadoun, David G. Kirkpatrick, and Teresa M. Przytycka. A simple parallel tree contraction algorithm. *Journal of Algorithms*, 10(2):287–302, 1989.

[CV88] Richard Cole and Uzi Vishkin. The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time. *Algorithmica*, 3:329–346, 1988.

[DK92] Arthur L. Delcher and Simon Kasif. Efficient parallel term matching and anti-unification. *Journal of Automated Reasoning*, 9(3):391–406, 1992.

[GCS94] Jeremy Gibbons, Wentong Cai, and David B. Skillicorn. Efficient parallel algorithms for tree accumulations. *Sci. Comput. Program.*, 23(1):1–18, 1994.

[GR89] Alan Gibbons and Wojciech Rytter. Optimal parallel algorithm for dynamic expression evaluation and context-free recognition. *Information and Computation*, 81(1):32–45, 1989.

[KME07] Kazuhiko Kakehi, Kiminori Matsuzaki, and Kento Emoto. Efficient parallel tree reductions on distributed memory environments. In *Computational Science - ICCS 2007, 7th International Conference, Beijing, China, May 27 - 30, 2007, Proceedings, Part II*, volume 4488 of *Lecture Notes in Computer Science*, pages 601–608. Springer, 2007.

[Mat07] Kiminori Matsuzaki. *Parallel Programming with Tree Skeletons*. PhD thesis, Graduate School of Information Science and Technology, The University of Tokyo, 2007.

[MHKT05] Kiminori Matsuzaki, Zhenjiang Hu, Kazuhiko Kakehi, and Masato Takeichi. Systematic derivation of tree contraction algorithms. *Parallel Processing Letters*, 15(3):321–336, 2005.

[MHT06] Kiminori Matsuzaki, Zhenjiang Hu, and Masato Takeichi. Towards automatic parallelization of tree reductions in dynamic programming. In *SPAA 2006: Proceedings of the 18th Annual ACM Symposium on Parallel Algorithms and Architectures, Cambridge, Massachusetts, USA, July 30 - August 2, 2006*, pages 39–48. ACM, 2006.

[MR85] Gary L. Miller and John H. Reif. Parallel tree contraction and its application. In *26th Annual Symposium on Foundations of Computer Science, 21-23 October 1985, Portland, Oregon, USA*, pages 478–489. IEEE, 1985.

[MW97] Ernst W. Mayr and Ralph Werchner. Optimal tree constraction and term matching on the hypercube and related networks. *Algorithmica*, 18(3):445–460, 1997.

[Rei93] John H. Reif, editor. *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers, 1993.

[Ski96] David B. Skillicorn. Parallel implementation of tree skeletons. *Journal of Parallel and Distributed Computing*, 39(2):115–125, 1996.