Parallel Tree-Contraction and Fibonacci Numbers *

Wojciech Plandowski Wojciech Rytter Tomasz Szymacha

Instytut Informatyki, Uniwersytet Warszawski,
ul. Banacha 2, 02-097 Warszawa, Poland

Key words: tree-contraction, parallel computations, expression evaluation

Abstract

We show a new property of **Fibonacci numbers** which is related to the analysis of a very simple and natural parallel **tree contraction** algorithm. We show that the size of the smallest tree which requires t contractions equals exactly the t-th *Fibonacci number*. This implies the **sharp bound** on the number of iterations of the tree contraction algorithm. We contribute also to combinatorics of trees.

Parallel evaluation of expressions and (related to it) *tree contraction* are fundamental techniques in the design of parallel algorithms. In this paper we consider a tree contraction algorithm related to the simultaneous substitutions algorithm of parallel evaluation of expressions given by *straight-line programs*, as given by Rytter in [Ry-90]. In this case the tree is the *computation graph* of the straight-line program, the parallel evaluation of the expression corresponds to the operations of compressing chains and (simulateneoulsy) removing edges leading to the leaves. We introduce a family of Fibonacci-type trees $\mathcal{F}_k$, which are the hardest trees for the algorithm. The importance of the algorithms analyzed here follows also from their relation to the parallel evaluation of straight line programs and recognition of context-free languages, see [GR-88], see [Ry-90] and [GR-88]. We believe that the exact analysis of the tree contraction algorithms is an interesting problem related to combinatorics of trees. Our model of parallel computation is a CREW Parallel Random Access Machine (CREW PRAM), see for example [GR-88]. The PRAM consists of a number of processors working synchronously and using a common global memory. Throughout the paper $\log_c n$ is to be understood as $\lceil \log_c n \rceil$, and $w/2$ as $\lfloor w/2 \rfloor$.

The algorithm SimSub works on straight-line programs representing the sequential evaluation of expression-trees bottom-up. A straight-line program P is a sequence of assignment statements, the i-th statement is of a form: $x_i = E_i$, where E_i is an expression involving some variables x_k with $k < i$.

*Supported by the grant KBN 8T11C01208

Example 2. The following sequence is an example of a straight-line program P:

$$x_1 = 1,\ x_2 = 2 * x_1,\ x_3 = 2,\ x_4 = 3,\ x_5 = 4,\ x_6 = x_2 + x_3,\ x_7 = x_4 + 1,\ x_8 = 2 * x_5$$
$$x_9 = 5,\ x_{10} = 2 * x_6 + 3 * x_7,\ x_{11} = x_8 * x_9,\ x_{12} = x_{10} + 2 * x_{11},\ x_{13} = 2 * x_{12} + 2.$$

The method of simultaneous substitutions generalizes the parallel assignment statement. The basic operation is the substitution: the variable x can be replaced by the right hand side of its equation, afterwards all possible local reductions take place. This operation is denoted by $subst(x)$. For example after applying $subst(x_7)$ to the right hand side of the equation for x_{10} in Example 1 we obtain $x_{10} = 2 * x_6 + 3 * x_4 + 3$. We refer the reader to [Ry-90] for details. The substitution of x is **safe** if the right hand side of the equation for x contains at most one variable. In our example initially substitutions involving the first six variables are safe.

Algorithm SimSub;
repeat
for each x_i **do in parallel**
apply all safe substitutions to the right hand side of the equation for x_i.
until x_n is computed.

Variables relevant to the computation of the output variable x_n are called **essential**, and only the equations for them are considered.

Example 2. Let us examine the history of the algorithm SimSub on our straight-line program P, from Example 1. After the first iteration we have:

$$x_1 = 1,\ x_4 = 3,\ x_5 = 4,\ x_6 = 2 * x_1 + 2,\ x_{10} = 2 * x_6 + 3 * x_4 + 3,\ x_{11} = 10 * x_5,$$
$$x_{12} = x_{10} + 2 * x_{11},\ x_{13} = 2 * x_{12} + 2.$$

After the second iteration:

$$x_1 = 1,\ x_{10} = 4 * x_1 + 16,\ x_{11} = 40,\ x_{12} = x_{10} + 2 * x_{11},\ x_{13} = 2 * x_{12} + 2.$$

After the third iteration we have

$$x_1 = 1,\ x_{12} = 4 * x_1 + 96,\ x_{13} = 2 * x_{12} + 2.$$

After the fourth iteration:

$$x_1 = 1,\ x_{13} = 8 * x_1 + 194.$$

The fifth iteration is the last one: $x_{13} = 202$.

The **computation graph** $T = graph(P)$ of the straight-line program P is the directed acyclic graph whose nodes are variables which are *essential*. The sons of a node x_i (variable) are variables occurring on the right hand side of the equation for x_i in a given moment. T contains only nodes reachable from x_n. Observe that the number of nodes of $graph(P)$ can be smaller than the size n of P. In the case of straight-line programs computing directly expressions T is a tree. The sequence of computation graphs for our example is shown in Fig. 1. We examine the effect of one iteration on the computation tree of the straight-line program P representing an expression. Let $T = graph(P)$. Now the variable is safe iff its corresponding node (in T) has outdegree at most one. Define $contract(T) = T' = graph(P')$, where P' is the program resulting from P after one iteration of the algorithm SimSub.

function contract(T);
{ edges are directed top-down}
for each directed edge $e = (v, w)$ of T **do in parallel**
  if w is a leaf **then** remove e **else**
  if w has degree 1 **then** delete e and create $(v, son(w))$.
return the tree consisting of all nodes accessible from the *root* of T

Algorithm Parallel_Tree_Contraction;
repeat
  T:= **contract**(T);
until $|T| = 1$.

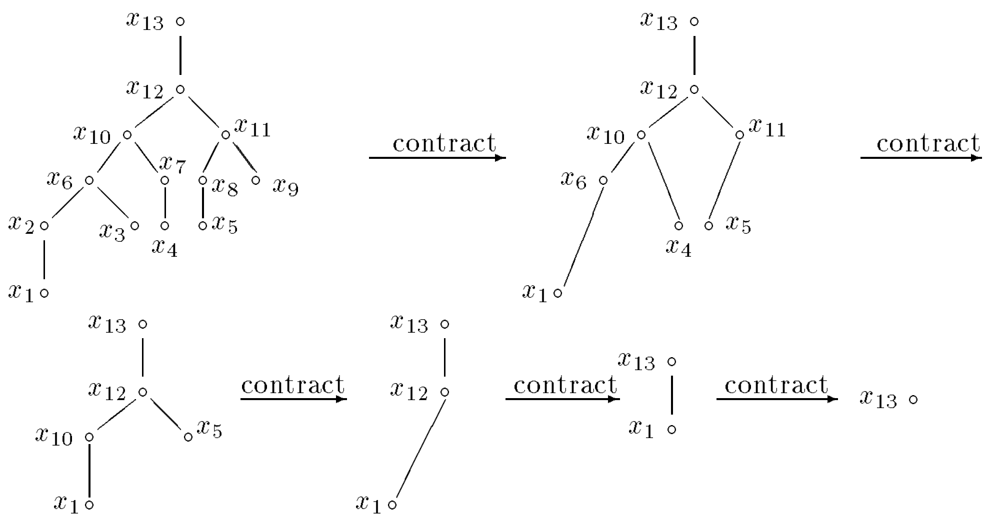

Figure 1: A history of the algorithm Parallel_Tree_Contraction: the sequence of contractions corresponding to iterations in Example 2.

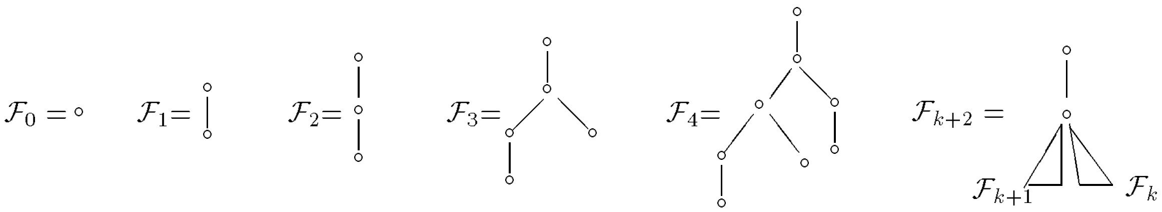

Figure 2: The structure of trees $\mathcal{F}_k$ and the trees $\mathcal{F}_k$, for $k \leq 4$. $\mathcal{F}_{k+1}$ and $\mathcal{F}_k$ have one node in common in $\mathcal{F}_{k+2}$, their roots are identified and the resulting node is a son of a newly created root.

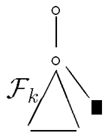

Figure 3: A tree of size $(Fib_k + 1)$ which needs $k+1$ contractions.

Denote by $TIME_{SimSub}(n)$ the maximal number of iterations of the algorithm Parallel_Tree_Contraction applied to trees with n nodes. It was proved in [Ry-90] that $TIME_{SimSub}(n) \leq \log_{3/2} n$. We improve this bound and give sharp bounds for the number of iterations.

Lemma 1 *For each t there is a tree of size Fib_t which requires t iterations.*

Proof:
A family of Fibonacci-like trees $\mathcal{F}_k$ is defined in Fig. 2 for $k \geq 0$.

Observe that $|\mathcal{F}_k| = Fib_k$ for $k \geq 0$. It is easy to see that $contract(\mathcal{F}_k) = \mathcal{F}_{k-1}$ for $k \geq 1$. Hence the tree $\mathcal{F}_k$ requires exactly k contractions. This completes the proof. □

The lower bound is proved using the concept of a static node. It is an artificial leaf whose ingoing edge can never be removed. If a tree T has exactly one static node then after several contractions it will become .

Denote by $TIME'(n)$ the biggest number of contractions needed to contract a tree with one static and n nonstatic nodes to the tree . For example the tree shown in Fig. 3 needs $k+1$ contractions and has size $Fib_k + 1$. The next obvious fact is useful later.

Fact 2 *Assume x is a node in a given tree T. Define T' and T_x as in Fig. 4. If t steps contract the tree T' to the tree of the form then after t contractions the root of T becomes a leaf or has a single son which is a leaf or has a single son which is a node of T_x.*

Lemma 3
(i) *if $n < Fib_k$ then $TIME_{SimSub}(n) < k$;*
(ii) *if $n \leq Fib_k$ then $TIME'(n) \leq k$.*

Proof:

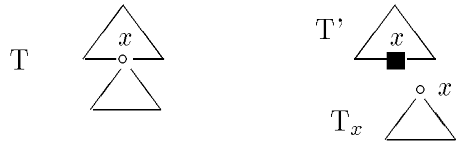

Figure 4: The definition of trees T' and T_x.

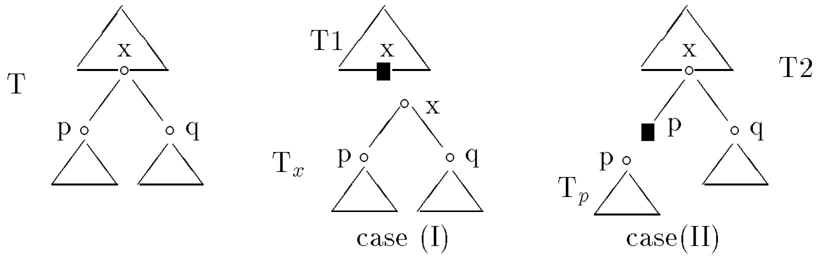

Figure 5: The tree T and its decompositions. $|T| < Fib_k$, $|T1| \leq Fib_{k-2}$, $|T_x| \geq Fib_{k-1}$, $|T_p| < Fib_{k-1}$.

The proof is by induction on k. For $k = 1, 2$ the thesis can be checked easily. We show the induction step. Assume that the claim is true for all k' smaller than a given $k > 2$.

Proof of (i)

Consider a tree T such that $|T| < Fib_k$. Take a lowest node x such that the tree T_x rooted at x satisfies $|T_x| \geq Fib_{k-1}$. Let p, q be the sons of x, see Fig. 5. (If x has exactly one son then let it be p).

Thus $|T_p|, |T_q| < Fib_{k-1}$. We are to prove that after reduction $k-1$ the root of T becomes a leaf. There are possible two cases.

Case (I). After the reduction $k-2$ the node x is evaluated. By the inductive assumption (ii) the tree $T1$ is contracted to . All the nodes of T_x are evaluated, because its root x is evaluated. Hence due to fact 2 the $(k-1)$-th reduction removes the last edge connecting the root of T with a leaf and the root becomes a leaf. The tree T is contracted to one node. The point (i) holds.

Case (II). The tree T_x needs at least $k-1$ contractions. For a tree R and a node v not contained in R we denote by $R \otimes v$ the tree composed from v and R in such a way that its root is v and the only son of v is the root of R. Then it is easy to see that one of the trees $T_p \otimes x$ or $T_q \otimes x$, say the first of them, needs at least $k-1$ contractions. Therefore from the inductive assumption (i) $|T_p \otimes x| \geq Fib_{k-1}$ (in that case $|T_p| = Fib_{k-1} - 1$). This implies that the tree $T2$ with a static node, see Fig. 5, is small. $|T2| \leq Fib_{k-2}$. (Recall that the static node does not contribute to the size of the tree). This implies, by the inductive assumption (ii), that the tree $T2$ is contracted to in $k-2$ contractions. All the nodes of T_p are evaluated after the reduction $k-2$ by the inductive assumption (i). Hence, by the same argument as in the former case, the root becomes a leaf. The whole tree is reduced to a single node.

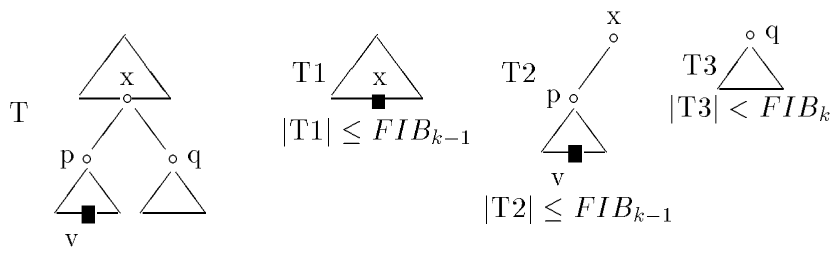

Figure 6: The decomposition of a tree T with the static node v.

Proof of (ii)

Consider a tree T with one static leaf v such that $|T| \le Fib_k$. Take the first node x on the path from the static node v to the root such that $|T_x| \ge Fib_{k-1}$. Let $T1$, $T2$ and $T3$ be the trees illustrated in Fig. 6. Then $|T1|, |T2| \le Fib_{k-1}$ and $|T3 \otimes x| \le Fib_k$.

Assume first that $|T3 \otimes x| = Fib_k$. Then the tree $T1$ consists of one static node x and the tree $T2$ consists of two vertices x and the static vertex v. Since $|T3| < Fib_k$ by the point (i) after the $(k-1)$-th reduction all vertices of the tree $T3$ are leaves. Hence, in the k-th reduction the tree T is contracted to the tree .

Assume now that $|T3 \otimes x| < Fib_k$. Then after the reduction $k-1$ the trees $T1$ and $T2$ are contracted to the tree , due to the inductive assumption (ii). All the nodes of $T3 \otimes x$, are leaves after the $(k-1)$-th reduction by the point (i). Consequently, after reduction $k-1$ there is no edge to any node in $T3$. The root points to its only son x' such that $x' = x$ or x' is in T_p. However each node of T_p except v points to v after the $(k-1)$-th reduction. Then in the k-th reduction the root points to the static node and the whole tree is contracted to the tree . Hence $TIME'(Fib_k) \le k$.

□

Theorem 4
(a) The size of the smallest tree requiring k contractions in the algorithm SimSub equals Fib_k.
(b) There is a constant c such that $\log_\Phi n \le TIME_{SimSub}(n) \le \log_\Phi n + c$

Proof: Direct consequence of Lemma 1 and Lemma 3. □

Generally the graph of the straight-line program has not to be a tree, as it was assumed in the former section. It may be any directed acyclic graph (dag, in short). Assume that the operations are $+$ and $*$. Assume also that initially each right hand side of the equation for each variable contains at most one operation. Outdegrees can be greater than two, and the outgoing edges (from $*$ nodes) are partitioned into two sets: left and right sons (which may overlap). This is because the multiplication nodes impose more complicated structures on their sons. The multiplication nodes correspond to variables x_i whose right sides are of the form:

$$(a_1x_1 + \ldots + a_kx_k) * (b_1y_1 + \ldots + b_ry_r) + c \ .$$

The set of sons of such a node x_i is partitioned into the set of left sons $\{x_1, \ldots, x_k\}$ and right sons $\{y_1, \ldots, y_r\}$.

If the degree of computed polynomials (assuming that constants correspond to input variables, and the leaves of the computation graph have degree one) is bounded by a polynomial then the straight-line program can be computed by an NC algorithm, see [GR-88] for the definition of NC. The algorithm SimSub can be also applied now (performing $O(\log n)$ iterations, each one consisting of all safe substitutions). The only difference is in the notion of safeness:
a substitution is called safe if the right hand side of the equation for the corresponding variable is of degree at most one (e.g. $x = x1 + 2 * x2 + x6$).

The safe substitutions correspond in a natural way to some matrix multiplications and this gives a simple parallel algorithm for the evaluation of straight-line programs, see [Ry-90].

Denote by SimSub' the version of the algorithm SimSub applied to straight-line programs with the second definition of safeness, see [Ry-90]. One iteration of this algorithm corresponds to the following operation contract' on the dag G.

Perform simultaneously the following actions:

- Every edge leading to a leaf is removed.
- Every edge leading from a node v to a plus-node w is replaced by edges leading from v directly to the sons of w.
- If after such actions a multiplication node has only left, or only right sons then it becomes a plus-node.

The iterative application of the operation contract' to a dag is called the dag-contraction algorithm. The tree contraction algorithm terminates iff the root is a leaf (has no outgoing edges). The number of iterations of this algorithm is the same as the number of iterations of the algorithm SimSub' applied to straight-line programs with the second definition of safeness.

Denote by $deg(P)$ and $deg(G)$ the degree of the straight-line program P and its corresponding computation graph G, respectively. The size $|P|$ and $|G|$ is the number of variables of P and nodes of G, respectively. The best bound on the number of iterations of the algorithm SimSub' was $\log_\alpha(deg(P)|P|) + c$, see [Ry-90], assuming that the tree contraction terminates in $\log_\alpha n$ iterations. We can now take $\alpha = \Phi$, due to Theorem 4, this gives the following result.

Theorem 5
(a) The dag-contraction algorithm has at most $\log_\Phi(deg(G) * |G|) + c_1$ *iterations on a given dag G;*
(b) the algorithm SimSub' makes at most $\log_\Phi(deg(P) * |P|) + c_2$ *iterations, where c_1 and c_2 are constants.*

References

[GR-89] Gibbons A, Rytter W: An optimal parallel algorithm for dynamic expression evaluation and its applications. Information and Computation 81, 32-45 (1989)

[GR-88] Gibbons A, Rytter W: Efficient parallel algorithms. Cambridge University Press, 1988

[Lu-81] de Luca A: A combinatorial property of the Fibonacci words, IPL 12:4 (1981) 193-195

[Ma-87] Mayr E: The dynamic tree expression problem, Report STAN-CS-87-1156, Stanford University (1987)

[MR-85] Miller G, Reif J: Parallel tree contraction and its application. 26th IEEE Symp. on Found. of Computer Science (1985) pp.478-489

[MRK-86] Miller G, Ramachandran V, Kaltofen E: Efficient parallel evaluation of straight line code and arithmetic circuits. Workshop on parallel algorithms, AWOC , Lect. Notes in Comp. Science (1986)

[Ry-90] Rytter W: On parallel computations of expressions and straight line programs, Computers and Artificial Intelligence 9:5 (1990) pp. 427-439

[Ry-88] Rytter W: On efficient parallel computations for some dynamic programming problems, TCS 59 (1988) 297-307