Parallel Processing Letters, Vol. 15, No. 3 (2005) 321–336

SYSTEMATIC DERIVATION OF TREE CONTRACTION ALGORITHMS*

KIMINORI MATSUZAKI, ZHENJIANG HU,
KAZUHIKO KAKEHI, and MASATO TAKEICHI
Graduate School of Information Science and Technology,
University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, 113-8656, Tokyo, JAPAN
{Kiminori_Matsuzaki,hu,kaz,takeichi}@mist.i.u-tokyo.ac.jp

Received 16 September 2004
Revised 31 October 2004
Communicated by Sergei Gorlatch

ABSTRACT

While tree contraction algorithms play an important role in efficient tree computation in parallel, it is difficult to develop such algorithms due to the strict conditions imposed on contracting operators. In this paper, we propose a systematic method of deriving efficient tree contraction algorithms from recursive functions on trees. We identify a general recursive form that can be parallelized into efficient tree contraction algorithms, and present a derivation strategy for transforming general recursive functions to the parallelizable form. We illustrate our approach by deriving a novel parallel algorithm for the maximum connected-set sum problem on arbitrary trees, the tree-version of the well-known maximum segment sum problem.

Keywords: Tree Contraction, Parallelization, Skeletal Parallelism, Rose Tree, Maximum Segment Sum Problem.

1. Introduction

Skeletal parallel programming [1,2] is an elegant model for developing efficient and correct parallel programs. Although many researchers have devoted themselves to the algorithmic skeletons on lists [3,4,5,6], not very many studies have been addressed to other datatypes such as trees and graphs.

Trees are important datatypes, widely used in representing hierarchical structures such as XML. There are two approaches to the parallel computation on trees; the first is the *divide and conquer* approach [7], and the second is the *tree contraction* approach [8,9,10,11]. The divide and conquer approach simply computes on each child tree independently, and its parallel cost is $O(h + w)$, where h denotes the height of a tree and w denotes the nodes' maximum number of children. Therefore, it may be very inefficient if the tree is ill-balanced or a node has too many children. By contrast, the tree contraction approach provides efficient parallel algorithms even for ill-balanced trees. The well-known algorithm, the *shunt contraction*, can run on binary trees in logarithmic time to their size. However, it requires the tree

*This work was partly supported by a PRESTO project of Japan Science and Technology Agency.

contracting operators that have to meet the closure property to be intelligently designed, which is known to be hard, and thus discourages programmers from using it.

Some attempts have been made on the formal specification for parallel tree algorithms. Gibbons et al. [12] and Skillicorn [13,14] defined five skeletons on binary trees and gave an efficient implementation of them based on the tree contraction algorithm. Skillicorn also showed the usefulness of these skeletons with several examples of the manipulation of structured documents [13,15,16]. Deldari et al. [17] designed a skeleton for the constructive solid geometry. Matsuzaki et al. [18] proposed a systematic method of composing efficient parallel programs in terms of the skeletons on binary trees. However, there have really been very few studies on formal derivation of parallel algorithms on general trees.

In this paper, we consider parallelization of a general tree recursive function, called (tree) reduction, which can concisely specify the computation of calculating a value through a bottom-up traversal of the tree. Informally, function f is a reduction, if it is defined in the following recursive form:

$$\begin{aligned} f\ (\mathit{RLeaf}\ a) &= k_1\ a \\ f\ (\mathit{RNode}\ b\ [t_1, t_2, \ldots, t_n]) &= k_2\ b\ [f\ t_1, f\ t_2, \ldots, f\ t_n]\ , \end{aligned}$$

where k_1 and k_2 are two functions. As discussed by Skillicorn [14,16], certain conditions on k_2 are necessary for the existence of efficient parallel algorithms. One sufficient condition proposed so far [14,16,19] is to define k_2 using associative operator $\oplus$ as follows.

$$\begin{aligned} &\mathit{reduce}\ (\oplus)\ (\mathit{RLeaf}\ a) = k_1'\ a \\ &\mathit{reduce}\ (\oplus)\ (\mathit{RNode}\ b\ [t_1, t_2, \ldots, t_n]) \\ &\quad = k_2'\ b \oplus \mathit{reduce}\ (\oplus)\ t_1 \oplus \mathit{reduce}\ (\oplus)\ t_2 \oplus \cdots \oplus \mathit{reduce}\ (\oplus)\ t_n \end{aligned}$$

This definition is easy to understand but lacks expressiveness. Consider the problem of XML serialization, which accepts an XML tree and returns its tagged-formatted string. We may solve it with the following recursive definition:

$$\begin{aligned} &\mathit{x2s}\ (\mathit{RLeaf}\ a) = a \\ &\mathit{x2s}\ (\mathit{RNode}\ b\ [t_1, t_2, \ldots, t_n]) = \mathit{tags}\ b \oplus (\mathit{x2s}\ t_1 +\!\!+ \mathit{x2s}\ t_2 +\!\!+ \cdots +\!\!+ \mathit{x2s}\ t_n) \\ &\qquad\qquad \textbf{where}\ \mathit{tags}\ b = (\text{“<”} +\!\!+ b +\!\!+ \text{“>”}, \text{“</”} +\!\!+ b +\!\!+ \text{“>”}) \\ &\qquad\qquad\qquad (s, e) \oplus t = s +\!\!+ t +\!\!+ e\ , \end{aligned}$$

where $+\!\!+$ is an infix-operator to concatenate two strings. It is not obvious, however, how to define *x2s* in terms of *reduce*, because we need to merge the two different binary operators, namely $\oplus$ and $+\!\!+$, into a single associative operator.

In this paper, we aim at a systematic method of parallelizing a class of useful reductions to ones that can be efficiently implemented by the tree contraction. Our method can deal with recursive definitions in which k_2 is defined using two binary operators. The contributions this paper makes can be summarized as follows:

- We give a new formalization of the condition for the shunt contraction (Theorem 1), which is more constructive in the sense that the tree contracting

operators can be derived from it. In addition, to eliminate the limitation where the shunt contraction is only applied to binary trees, we show a representation of rose trees (trees whose nodes can have an arbitrary number of children) by binary trees so that the shunt contraction can be applied.

- We not only recognize the importance of distributivity in the derivation of the tree contraction algorithms, but also give an extension of distributivity that is suited to systematic derivation with generalization and context-preserving transformation. We identify a general recursive form that can be parallelized (Theorem 2), and highlight a derivation strategy for transforming general recursive functions to the parallelizable form.
- We demonstrate the effectiveness of our approach by deriving an efficient parallel program for the tree version of the maximum segment sum problem [20]. Several studies have been done on the parallelization of the problem: on lists [3,21], on 2-dimensional arrays [22], and on binary trees [18]. To the best of our knowledge, this is the first derivation of the parallel program for rose trees, the most complex data structure ever.

This paper is organized as follows. After reviewing notational conventions and datatypes, we show how an arbitrary tree is arranged in the form of a binary tree in Section 2. We formalize the conditions for the tree contraction in a more constructive way in Section 3, and give a property that is an extension of distributivity in Section 4. In Section 5, we define parallelizable reductions on rose trees, and show how these reductions are parallelized. We propose a strategy for systematic parallelization and demonstrate the effectiveness of our approach with a non-trivial example in Section 6, and conclude in Section 7.

2. Preliminaries

2.1. Functions and Operators

Function application is denoted by a space and the argument may be written without brackets. Thus $f\ a$ means $f(a)$. Functions are curried, and the function application associates to the left. Thus $f\ a\ b$ means $(f\ a)\ b$. The function application binds stronger than any other operator, so $f\ a \oplus b$ means $(f\ a) \oplus b$, but not $f\ (a \oplus b)$. Infix binary operators will be denoted by $\oplus$, $\otimes$, and their units are written as $\iota_\oplus$, $\iota_\otimes$, respectively.

2.2. Datatypes

The *cons list* is constructed with an empty list or by adding an element to a list. The datatype for lists where every element has type α is defined as follows.

$$\text{data } \mathit{List}\ \alpha = \mathit{Nil} \mid \mathit{Cons}\ \alpha\ (\mathit{List}\ \alpha)$$

We may use abbreviations, i.e., $[\alpha]$ for datatype $\mathit{List}\ \alpha$, $[\,]$ for Nil, and $(a : as)$ for $\mathit{Cons}\ a\ as$.

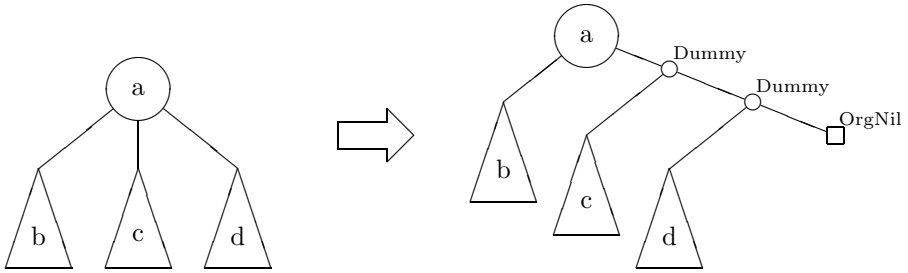

Fig. 1. Local rearrangement from a rose tree into a binary tree.

A binary tree is a tree whose internal nodes have exactly two children. The datatype for binary trees where every leaf has type α and every internal node has type β is defined as follows.

$$\text{data } \mathit{BTree}\ \alpha\ \beta = \mathit{Leaf}\ \alpha \mid \mathit{Node}\ \beta\ (\mathit{BTree}\ \alpha\ \beta)\ (\mathit{BTree}\ \alpha\ \beta)$$

A rose tree is a tree whose internal nodes have an arbitrary number of children. The datatype for rose trees where every leaf has type α and every internal node has type β is defined using a list as follows.

$$\text{data } \mathit{RTree}\ \alpha\ \beta = \mathit{RLeaf}\ \alpha \mid \mathit{RNode}\ \beta\ [\mathit{RTree}\ \alpha\ \beta]$$

2.3. Representation of Rose Trees

Since the shunt contraction algorithm only accepts binary trees, rose trees ought to be held in the shape of binary trees. In this paper, we will use the arrangement (representation) in Fig. 1. This arrangement turns each leaf and internal node of a rose tree into a leaf and the root node of the corresponding subtree in the binary tree, respectively. Some dummy nodes are inserted into this binary tree to unroll the children and to represent the children's end. This is almost the same arrangement as in [14], and there have been some discussions about the implementation of the tree contraction algorithm on these arranged binary trees.

To formally define the arrangement, we first define two new types.

$$\text{data } \mathit{R2BLeaf}\ \alpha\ = \mathit{OrgLeaf}\ \alpha \mid \mathit{OrgNil}$$
$$\text{data } \mathit{R2BNode}\ \beta = \mathit{OrgNode}\ \beta \mid \mathit{Dummy}$$

R2BLeaf represents the types of leaves in the binary tree, and is constructed with the leaf in the rose tree (*OrgLeaf*) or the sentinel for the end of the children (*OrgNil*). *R2BNode* represents the types of internal nodes in the binary tree, and is constructed with the internal node in the rose tree (*OrgNode*) or the dummy node inserted to expand the children (*Dummy*). The function *r2b* which performs this arrangement can be formally defined using auxiliary function $r2b'$ as follows.

$$r2b :: \mathit{RTree}\ \alpha\ \beta \rightarrow \mathit{BTree}\ (\mathit{R2BLeaf}\ \alpha)\ (\mathit{R2BNode}\ \beta)$$

$$
\begin{aligned}
&r2b\ (RLeaf\ a) &&= Leaf\ (OrgLeaf\ a)\\
&r2b\ (RNode\ b\ (x : xs)) &&= Node\ (OrgNode\ b)\ (r2b\ x)\ (r2b'\ xs)\\
&r2b'\ [\,] &&= Leaf\ OrgNil\\
&r2b'\ (x : xs) &&= Node\ Dummy\ (r2b\ x)\ (r2b'\ xs)
\end{aligned}
$$

We briefly analyze the number of resulting nodes in the binary trees after the above transformation. Let n_l be the number of leaves, and n_{in} be the number of internal nodes in an input rose tree. The binary tree transformed from the rose tree has $2n_l + 2n_{in} - 1$ nodes. Consequently, the number of nodes in transformed trees is no more than twice of that in original trees, and this guarantees the logarithmic parallel cost for logarithmic algorithms on these transformed trees.

3. Tree Contraction Algorithm and its Derivation

The tree contraction algorithms are efficient parallel algorithms to trees. Of the tree contraction algorithms, the shunt contraction [8] is widely known as a simple and efficient algorithm on EREW PRAM. The shunt contraction algorithm accepts binary trees, and reduces them with simultaneous applications of two symmetric operations, namely ContractL and ContractR. A ContractL/ContractR operation removes a left/right leaf and its parent from a binary tree.

In the following, we assume reductions on binary trees are defined as follows.

$$
\begin{aligned}
&f\ (Leaf\ a) &&= k_1\ a\\
&f\ (Node\ b\ t_1\ t_2) &&= k_2\ b\ (f\ t_1)\ (f\ t_2)
\end{aligned}
$$

The shunt contraction algorithm imposes certain conditions on the functions. A sufficient condition is the existence of an indexed set of functions G satisfying the following conditions.

- For every internal node $Node\ b\ t_1\ t_2$, function $\lambda xy.k_2\ b\ x\ y$ is drawn from G.
- For any functions g_i, g_j in G, and values l and r, two functions $\lambda xy.g_i\ l\ (g_j\ x\ y)$ and $\lambda xy.g_i\ (g_j\ x\ y)\ r$ are in G.

If all the functions in G can be applied in a constant time, and the indices of the composed functions are computed in a constant time, we can guarantee the overall logarithmic parallel cost of the shunt contraction algorithm. This condition is, however, too abstract for the programmers to utilize the tree contraction, since how to find such a set of functions G with suitable indices is not shown.

To enable systematic derivation, we introduce the idea of parametrized functions. We use notation $G[a]$ for the function embodied from the set of parametrized functions G with parameter a. For example, if the set of parametrized functions is given as $G[a] = \lambda xy.\ a + x + y$, then functions $G[1] = \lambda xy.\ 1 + x + y$ and $G[2] = \lambda xy.\ 2 + x + y$ are the embodiments with 1 and 2, respectively.

Let us now restrict the indexed set of functions to be the set of parametrized functions. Although some algorithms, in which different functions are applied to internal nodes, may be unacceptable under this restriction, numerous tree reduction algorithms can be dealt with. With the notation of parametrized functions, a sufficient condition for the shunt contraction is given by the following theorem.

(i) Number the leaves left to right beginning at 0.
(ii) Initialize every leaf and internal node by applying ψ_1 and ψ_2, respectively.
(iii) Iterate until a node remains.
  (a) For every left leaf whose index is even, perform ContractL. If the other child is a leaf apply a function embodied from G with parent's value n, or otherwise apply ϕ_L.
  (b) For every right leaf whose index is even and not involved in the previous step, perform ContractR. If the other child is a leaf apply $G[n]$, or otherwise apply ϕ_R.
  (c) Renumber the leaves by dividing their indices by two and rounding down.

Fig. 2. Shunt contraction algorithm based on parametrized functions

Theorem 1 If there are a set of parametrized functions G, and three functions ψ_2, ϕ_L and ϕ_R such that the following conditions are satisfied, then we can implement the algorithm with the shunt contraction algorithm.

- For every internal node $Node\ b\ t_1\ t_2$, function $\lambda xy.k_2\ b\ x\ y$ is semantically equivalent to function $G[\psi_2\ b]$.
- For any a_1, a_2, l and r, the following equations hold.

$$\lambda xy.G[a_1]\ l\ (G[a_2]\ x\ y) = \lambda xy.G[\phi_L\ a_1\ l\ a_2]\ x\ y$$
$$\lambda xy.G[a_1]\ (G[a_2]\ x\ y)\ r = \lambda xy.G[\phi_R\ a_1\ r\ a_2]\ x\ y$$

Proof. We can prove the parametrized functions set G satisfies the conditions of the shunt contraction by regarding the parameter of G as the index. If a tree algorithm meets these conditions, we can utilize the shunt contraction algorithm as shown in Fig. 2, where function ψ_1 is equal to k_1. □

Therefore, we only have to derive the set of parametrized functions G and functions ψ_2, ϕ_L and ϕ_R from the definition of k_2. To demonstrate how Theorem 1 works, let us illustrate it with a very simple program.

Example 1 A recursive program that computes the sum of values for all nodes is given as follows.

$$sumtree\ (Leaf\ a) = a$$
$$sumtree\ (Node\ b\ t_1\ t_2) = b + sumtree\ t_1 + sumtree\ t_2$$

An adequate definition of the set of parametrized functions G is given with parameter a as $G[a] = \lambda xy.a + x + y$. From the definition above, the initializing functions are $\psi_1 = id$ and $\psi_2 = id$, where id is the identity function. The contracting operations ϕ_L and ϕ_R become $\phi_L\ a_1\ l\ a_2 = a_1 + l + a_2$ and $\phi_R\ a_1\ r\ a_2 = a_1 + r + a_2$. With Theorem 1, we can parallelize *sumtree* with the tree contraction algorithm as shown in Fig. 2 using these functions. □

4. Extension of Distributive Law

Before discussing the parallelization of reductions, let us now discuss a generalization of the distributive law. It is well known that associativity and distributivity play important roles in parallelizing programs. For example, the associativity and the distributivity of $\times$ and $+$ enable us to simplify the expression $1+2\times(3+4\times x)$ into $7+8\times x$. Borrowing the idea of *contexts* or normal forms from [23], we define the characteristic of simplification over two operators as an extension of distributivity.

Definition 1 Let operator $\otimes$ be associative. The function defined with two operators, $\otimes$ and $\oplus$, is said to be in *normal form*, if it is written as $\lambda x.a \oplus (b \otimes x \otimes c)$, where a, b, and c are constants. □

Definition 2 Operator $\otimes$ is said to be *extended-distributive* over $\oplus$, if the normal form is preserved under function composition. In other words, there are appropriate functions p_1, p_2, and p_3, and for any a_1, b_1, c_1, a_2, b_2, and c_2, the following equation holds.

$$(\lambda x.a_1 \oplus (b_1 \otimes x \otimes c_1)) \circ (\lambda x.a_2 \oplus (b_2 \otimes x \otimes c_2)) = \lambda x.A \oplus (B \otimes x \otimes C)$$
$$\textbf{where } A = p_1\ (a_1, b_1, c_1, a_2, b_2, c_2)$$
$$B = p_2\ (a_1, b_1, c_1, a_2, b_2, c_2)$$
$$C = p_3\ (a_1, b_1, c_1, a_2, b_2, c_2)$$

Functions p_1, p_2, and p_3 are called *characteristic functions*. □

Although the definition of extended-distributivity is a little complicated, it has an advantage of many applications. We can make uniform use of this property for the associative operator, the distributive operators, or other operators, as demonstrated in the following examples. In Example 4, we also demonstrate how to derive characteristic functions from the definition of operators.

Example 2 Extended-distributivity can replace associativity. Let operator $\oplus$ be the same as associative operator $\otimes$. Then, $\otimes$ is extended-distributive over $\oplus$ ($= \otimes$) and the characteristic functions are as follows.

$$p_1\ (a_1, b_1, c_1, a_2, b_2, c_2) = a_1 \otimes b_1 \otimes a_2 \otimes b_2$$
$$p_2\ (a_1, b_1, c_1, a_2, b_2, c_2) = \iota_\otimes$$
$$p_3\ (a_1, b_1, c_1, a_2, b_2, c_2) = c_2 \otimes c_1$$ □

Example 3 Extended-distributivity is a generalization of the distributive law. Let two operators $\otimes$ and $\oplus$ constitute a ring, that is, let $\oplus$ be associative and $\otimes$ be not only associative but also distributive over $\oplus$. Then $\otimes$ is extended-distributive over $\oplus$ and the characteristic functions are as follows.

$$p_1\ (a_1, b_1, c_1, a_2, b_2, c_2) = a_1 \oplus (b_1 \otimes a_2 \otimes c_1)$$
$$p_2\ (a_1, b_1, c_1, a_2, b_2, c_2) = b_1 \otimes b_2$$
$$p_3\ (a_1, b_1, c_1, a_2, b_2, c_2) = c_2 \otimes c_1$$ □

To validate the extended-distributivity and derive the characteristic functions, we calculate two expressions E_1 and E_2 defined as

$$\begin{aligned}\lambda x.\ E_1 &= (\lambda x.a_1 \oplus (b_1 \otimes x \otimes c_1)) \circ (\lambda x.a_2 \oplus (b_2 \otimes x \otimes c_2)) \\ &= \lambda x.\ a_1 \oplus (b_1 \otimes (a_2 \oplus (b_2 \otimes x \otimes c_2)) \otimes c_1)\end{aligned}$$

$$\lambda x.\ E_2 = \lambda x.\ A \oplus (B \otimes x \otimes C)$$

and verify that they are the same by substituting proper expressions for the capital parameters in E_2. To demonstrate the derivation of characteristic functions, we show that the operators $+\!\!+$ and $\oplus$ in the definition of *x2s* in the introduction satisfy extended-distributivity and derive the characteristic functions.

Example 4 Let operator $\oplus$ be defined with the associative operator $+\!\!+$ as $(s, e) \oplus t = s+\!\!+t+\!\!+e$. The operator $+\!\!+$ is not distributive over $\oplus$, since $((s, e) \oplus (s', e'))+\!\!+t$ raises a type error. The operator $\oplus$ is not distributive over $+\!\!+$ either, i.e. $(s, e) \oplus (x+\!\!+y) \neq ((s, e) \oplus x)+\!\!+((s, e) \oplus y)$, which is easily seen with the simple calculation below.

$$\begin{aligned}(s, e) \oplus (x+\!\!+y) &= s+\!\!+x+\!\!+y+\!\!+e\\ ((s, e) \oplus x)+\!\!+((s, e) \oplus y) &= s+\!\!+x+\!\!+e+\!\!+s+\!\!+y+\!\!+e\end{aligned}$$

To validate extended-distributivity, we first expand the two expressions E_1 and E_2.

$$\begin{aligned}E_1 &= (s_1, e_1) \oplus (t_1+\!\!+((s_2, e_2) \oplus (t_2+\!\!+x+\!\!+t'_2))+\!\!+t'_1)\\ &= (s_1, e_1) \oplus (t_1+\!\!+s_2+\!\!+t_2+\!\!+x+\!\!+t'_2+\!\!+e_2+\!\!+t'_1)\\ &= s_1+\!\!+t_1+\!\!+s_2+\!\!+t_2+\!\!+x+\!\!+t'_2+\!\!+e_2+\!\!+t'_1+\!\!+e_1\\ E_2 &= (S, E) \oplus (T+\!\!+x+\!\!+T')\\ &= S+\!\!+T+\!\!+x+\!\!+T'+\!\!+E\end{aligned}$$

From the calculation above, the correspondences of capital variables are,

$$\begin{aligned}S+\!\!+T &= s_1+\!\!+t_1+\!\!+s_2+\!\!+t_2 \text{ , and}\\ T'+\!\!+E &= t'_2+\!\!+e_2+\!\!+t'_1+\!\!+e_1 \text{ .}\end{aligned}$$

There are many solutions to the equations above, and one of those is as follows, which can also be considered as a set of characteristic functions.

$$\begin{aligned}p_1\ ((s_1, e_1), t_1, t'_1, (s_2, e_2), t_2, t'_2) &= (S, E) = (s_1+\!\!+t_1+\!\!+s_2+\!\!+t_2, t'_2+\!\!+e_2+\!\!+t'_1+\!\!+e_1)\\ p_2\ ((s_1, e_1), t_1, t'_1, (s_2, e_2), t_2, t'_2) &= T = [\,]\\ p_3\ ((s_1, e_1), t_1, t'_1, (s_2, e_2), t_2, t'_2) &= T' = [\,]\end{aligned}$$

We can show extended-distributivity and derive the characteristic functions for general cases where $\oplus$ is defined similarly with associative operator $\otimes$. □

If operator $\otimes$ is also commutative, then we can simplify the definitions of the normal form and extended-distributivity. The normal form $\lambda xy.a \oplus (b \otimes x \otimes c)$ can be simplified to $\lambda xy.a \oplus (b' \otimes x)$ by swapping x and c, and substituting b' for $b \otimes c$. Extended-distributivity can be defined in terms of this form, and we say $\otimes$ is extended-distributive over $\oplus$ if there are appropriate functions p_1 and p_2 such that for any a_1, b_1, a_2, and b_2 the following equation holds. The characteristic functions are minimized into two functions p_1 and p_2 in this case.

$$\begin{aligned}(\lambda x.a_1 \oplus (b_1 \otimes x)) \circ (\lambda x.a_2 \oplus (b_2 \otimes x)) &= \lambda x.A \oplus (B \otimes x)\\ &\quad \textbf{where } A = p_1\ (a_1, b_1, a_2, b_2)\\ &\qquad\quad B = p_2\ (a_1, b_1, a_2, b_2)\end{aligned}$$

5. Parallelizable Reduction

In this section, we present a class of reductions that can be systematically parallelized based on the tree contraction algorithm. Reductions are a class of recursive computations that collapse a rose tree into a single value in a bottom-up manner, and the general definition of them is as follows.

$$\begin{aligned} f\ (\mathit{RLeaf}\ a) &= k_1\ a \\ f\ (\mathit{RNode}\ b\ [t_1, t_2, \ldots, t_n]) &= k_2\ b\ [f\ t_1, f\ t_2, \ldots, f\ t_n] \end{aligned}$$

Definition 3 Let $\otimes$ be an associative operator. A function is said to be a *parallelizable reduction*, if the function is defined in the following form.

$$\begin{aligned} f\ (\mathit{RLeaf}\ a) &= k_1\ a \\ f\ (\mathit{RNode}\ b\ [t_1, t_2, \ldots, t_n]) &= k_2\ b\ \oplus (f\ t_1 \otimes f\ t_2 \otimes \cdots \otimes f\ t_n) \end{aligned}$$

We can rephrase this using auxiliary function f' more formally.

$$\begin{aligned} f\ (\mathit{RLeaf}\ a) &= k_1\ a \\ f\ (\mathit{RNode}\ b\ ts) &= k_2\ b \oplus f'\ ts \\ f'\ [\,] &= \iota_\otimes \\ f'\ (t : ts) &= f\ t \otimes f'\ ts \end{aligned}$$

□

A parallelizable reduction is defined in two steps for each node. First, the siblings are collapsed with associative operator $\otimes$, which is the same operation as the reduction on lists. Then, another operator $\oplus$ merges the result of children and the parent value. We can write many reductions in this form, for example, the XML serialization in the introduction, the sum of values for all nodes, and several algorithms for structured documents [16].

In the following, we will demonstrate that parallelizable reductions can be efficiently computed with the tree contraction algorithm on arranged binary trees. Let the set of parametrized functions G be defined as: $G[(a, b, c)] = \lambda xy.a \oplus (b \otimes x \otimes y \otimes c)$. Using embodiments of this set of parametrized functions G, we can describe new function h on the arranged binary trees as follows.

$$\begin{aligned} h\ (\mathit{Leaf}\ (\mathit{OrgLeaf}\ a)) &= k_1\ a \\ h\ (\mathit{Leaf}\ \mathit{OrgNil}) &= \iota_\otimes \\ h\ (\mathit{Node}\ (\mathit{OrgNode}\ b)\ t_1\ t_2) &= G[(k_2\ b, \iota_\otimes, \iota_\otimes)]\ (h\ t_1)\ (h\ t_2) \\ h\ (\mathit{Node}\ \mathit{Dummy}\ t_1\ t_2) &= G[(\iota_\oplus, \iota_\otimes, \iota_\otimes)]\ (h\ t_1)\ (h\ t_2) \end{aligned}$$

Let us first prove the equivalence of h on the arranged binary trees.

Lemma 1 Function h defined above satisfies $h \circ \mathit{r2b} = f$, $h \circ \mathit{r2b}' = f'$.

Proof. We can prove this lemma by induction over the rose tree: base cases for $\mathit{RLeaf}\ a$ and $[\,]$, and inductive cases for $\mathit{RNode}\ b\ (t : ts)$ and $(t : ts)$, respectively. □

Next, let us prove that the set of parametrized functions G satisfies the conditions for the tree contraction algorithm.

Lemma 2 Let $\otimes$ be an associative operator and be distributive over $\oplus$ with characteristic functions p_1, p_2, and p_3. Then, for any parameters a_1, b_1, c_1, a_2, b_2, c_2, and values l and r, the following two equations

$$\lambda xy.G[(a_1,b_1,c_1)]\ l\ (G[(a_2,b_2,c_2)]\ x\ y) = \lambda xy.G[\phi_L\ (a_1,b_1,c_1)\ l\ (a_2,b_2,c_2)]\ x\ y$$
$$\lambda xy.G[(a_1,b_1,c_1)]\ (G[(a_2,b_2,c_2)]\ x\ y)\ r = \lambda xy.G[\phi_R\ (a_1,b_1,c_1)\ r\ (a_2,b_2,c_2)]\ x\ y$$

hold for appropriate functions ϕ_L and ϕ_R.

Proof. We define the two functions using $\otimes$, p_1, p_2, and p_3 as follows.

$$\begin{aligned}
\phi_L\ (a_1,b_1,c_1)\ l\ (a_2,b_2,c_2) &= (p_1\ tup_L, p_2\ tup_L, p_3\ tup_L) \\
&\quad \textbf{where}\ tup_L = (a_1, b_1 \otimes l, c_1, a_2, b_2, c_2)\ , \\
\phi_R\ (a_1,b_1,c_1)\ r\ (a_2,b_2,c_2) &= (p_1\ tup_R, p_2\ tup_R, p_3\ tup_R) \\
&\quad \textbf{where}\ tup_R = (a_1, b_1, r \otimes c_1, a_2, b_2, c_2)\ .
\end{aligned}$$

We can verify these two equations with simple calculations. □

Theorem 2 Function f defined in Definition 3 can be parallelized with the tree contraction algorithm on binary trees as arranged in Section 2, if operator $\otimes$ is associative and extended-distributive over $\oplus$.

Proof. Let the characteristic functions of extended-distributivity be p_1, p_2, and p_3. We can construct the initialize functions ψ_1 and ψ_2, the contracting operations ϕ_L and ϕ_R, and the set of functions G in the following way. In the rest of this paper, due to space limitations, we will place the definitions of ψ_1 and ψ_2 side by side.

$$\begin{aligned}
&\psi_1\ (OrgLeaf\ a) = k_1\ a && \psi_2\ (OrgNode\ b) = (k_2\ b, \iota_\otimes, \iota_\otimes) \\
&\psi_1\ OrgNil\ \ \ = \iota_\otimes && \psi_2\ Dummy\ \ \ = (\iota_\oplus, \iota_\otimes, \iota_\otimes) \\
&\phi_L\ (a_1,b_1,c_1)\ l\ (a_2,b_2,c_2) = (p_1\ tup_L, p_2\ tup_L, p_3\ tup_L) \\
&\qquad \textbf{where}\ tup_L = (a_1, b_1 \otimes l, c_1, a_2, b_2, c_2) \\
&\phi_R\ (a_1,b_1,c_1)\ r\ (a_2,b_2,c_2) = (p_1\ tup_R, p_2\ tup_R, p_3\ tup_R) \\
&\qquad \textbf{where}\ tup_R = (a_1, b_1, r \otimes c_1, a_2, b_2, c_2) \\
&G[(a,b,c)] = \lambda xy.a \oplus (b \otimes x \otimes y \otimes c)
\end{aligned}$$

It follows from Lemmas 1 and 2 that the theorem holds. □

To illustrate an application of this theorem, let us derive a parallel algorithm from the definition of *x2s* in the introduction.

Example 5 Function *x2s* can be computed in parallel because operator $+\!\!+$ is associative and extended-distributive over $\oplus$ as mentioned in Example 4. We can derive a parallel program according to Theorem 2 by utilizing the result of Example 4, and the derived program is as follows.

$$\begin{aligned}
&\psi_1\ (OrgLeaf\ a) = a && \psi_2\ (OrgNode\ b) = (tags\ b, [\,], [\,]) \\
&\psi_1\ OrgNil\ \ \ = [\,] && \psi_2\ Dummy\ \ \ = (([\,],[\,]),[\,],[\,]) \\
&\phi_L\ ((s_1,e_1),t_1,t_1')\ l\ ((s_2,e_2),t_2,t_2') \\
&\qquad = ((s_1 +\!\!+ t_1 +\!\!+ l +\!\!+ s_2 +\!\!+ t_2, t_2' +\!\!+ e_2 +\!\!+ t_1' +\!\!+ e_1), [\,], [\,]) \\
&\phi_R\ ((s_1,e_1),t_1,t_1')\ r\ ((s_2,e_2),t_2,t_2') \\
&\qquad = ((s_1 +\!\!+ t_1 +\!\!+ s_2 +\!\!+ t_2, t_2' +\!\!+ e_2 +\!\!+ r +\!\!+ t_1' +\!\!+ e_1), [\,], [\,]) \\
&G[((s,e),t,t')] = \lambda xy.s +\!\!+ t +\!\!+ x +\!\!+ y +\!\!+ t' +\!\!+ e
\end{aligned}$$

□

If operator $\otimes$ is not only associative but also commutative, then we can derive a simpler parallel program as the following corollary shows.

Corollary 1 Let operator $\otimes$ in a parallelizable reduction be both associative and commutative. If operator $\otimes$ is also extended-distributive over $\oplus$ with characteristic functions p_1 and p_2, we can parallelize the reduction with the tree contraction algorithm with the following functions. The two contracting operations, ϕ_L and ϕ_R, have the same definition, namely ϕ.

$$\begin{aligned}
&\psi_1\ (\mathit{OrgLeaf}\ a) = k_1\ a && \psi_2\ (\mathit{OrgNode}\ b) = (k_2\ b, \iota_\otimes)\\
&\psi_1\ \mathit{OrgNil} = \iota_\otimes && \psi_2\ \mathit{Dummy} = (\iota_\oplus, \iota_\otimes)\\
&\phi\ (a_1, b_1)\ x\ (a_2, b_2) = (p_1\ (a_1, b_1 \otimes x, a_2, b_2),\ p_2\ (a_1, b_1 \otimes x, a_2, b_2))\\
&G[(a, b)] = \lambda xy.a \oplus (b \otimes x \otimes y)
\end{aligned}$$

□

6. Parallelization Strategy

Although we have extended-distributivity and parallelizable reductions in hand, users' programs may not be exactly compatible with them. Even so, we can still derive parallel programs systematically with the following strategy.

(i) *Write specification*: In the first step, we write the specification as a recursive function in the form of parallelizable reductions. In this step, the operators used in the function do not need to be associative or extended-distributive. We derive the program in the form of parallelizable reductions by applying calculational techniques such as tupling or normalization of conditions.

(ii) *Derive associative operator*: In the second step, we derive an associative operator for $\otimes$, by applying the parallelization techniques that have been proposed for lists, for example, the fusion and tupling technique proposed by Hu et al. [21] or the context preservation technique proposed by Chin et al. [23].

(iii) *Derive extended-distributive operator*: In the third step, we derive operator $\oplus$ such that the operator $\otimes$ is extended-distributive over $\oplus$. We derive such an operator by iterative generalization and verification. To avoid inconsistency over the $\otimes$, we only generalize the definition of $\oplus$ for the left argument.

(iv) *Derive parallel program*: In the final step, we derive the contracting operations based on Theorem 2, and do some optimizations if possible.

In the following, we illustrate the capability of our parallelization strategy, by demonstrating the derivation of an efficient parallel program for the maximum connected-set sum problem on trees with arbitrary shapes, which is the tree version of the maximum segment sum problem [20]. This problem involves finding the maximum sum of all connected sets. A connected-set of a tree is a set of nodes where every two nodes are connected or connected through the nodes in the set.

6.1. Write specification

We first write a recursive function for the problem. For the maximum connected-set sum problem, we can write a program using the dynamic programming technique, where the following two values are computed for each subtree.

- r: The maximum sum of all connected sets that include the root of the subtree. We can compute this value by adding the value of the root node to the sum of all positive r values of the root's immediate subtrees.
- s: The maximum sum of all connected sets that do not include the root of the subtree. We can compute this value by selecting the maximum r and s values of the root's immediate subtrees.

With this idea, we can define the following function, where binary operator $\uparrow$ returns the larger value.

$$
\begin{aligned}
&\mathit{mcs}\ t && = \mathbf{let}\ (r,s) = \mathit{mcs}'\ t\ \mathbf{in}\ r \uparrow s \\
&\mathit{mcs}'\ (\mathit{RLeaf}\ a) && = (a,0) \\
&\mathit{mcs}'\ (\mathit{RNode}\ b\ [t_1,t_2,\ldots,t_n]) && = b \oplus (g\ (\mathit{mcs}'\ t_1) \otimes g\ (\mathit{mcs}'\ t_2) \otimes \cdots \otimes g\ (\mathit{mcs}'\ t_n)) \\
& && \quad \mathbf{where}\ b \oplus (r,s) = (b+r,s) \\
& && \qquad g\ (r,s) = (r \uparrow 0, r \uparrow s) \\
& && \qquad (r,s) \otimes (r',s') = (r+r', s \uparrow s')
\end{aligned}
$$

The function above is not a parallelizable reduction, since there are extra calls of g for each subtree. To obtain a parallelizable reduction, we fuse functions g and mcs' and introduce function $\mathit{mcs2}'$ defined as $\mathit{mcs2}'\ t = g\ (\mathit{mcs}'\ t)$ and operator $\oplus'$ defined as $b \oplus' t = g\ (b \oplus t)$, that is, $b\ \oplus'\ (r,s) = ((b+r) \uparrow 0, (b+r) \uparrow s)$. For the top-level call of $\mathit{mcs2}'$, we select the second value with function snd. We then obtain the following definition, which is a parallelizable reduction.

$$
\begin{aligned}
&\mathit{mcs2}\ t && = \mathit{snd}\ (\mathit{mcs2}'\ t) \\
&\mathit{mcs2}'\ (\mathit{RLeaf}\ a) && = (a \uparrow 0, a \uparrow 0) \\
&\mathit{mcs2}'\ (\mathit{RNode}\ b\ [t_1,t_2,\ldots,t_n]) && = b \oplus' (\mathit{mcs2}'\ t_1 \otimes \mathit{mcs2}'\ t_2 \otimes \cdots \otimes \mathit{mcs2}'\ t_n)
\end{aligned}
$$

6.2. Derive associative operator

The operator $\otimes$ is fortunately associative and commutative, by reason of the associativity and commutativity of $\uparrow$ and $+$. The unit of $\otimes$ is $\iota_\otimes = (0, -\infty)$.

6.3. Derive extended-distributive operator

To validate whether operator $\otimes$ is extended-distributive over operator $\oplus'$, we match the following expression E_1 to E_2. In the following, we may denote a tuple as a column vector for readability.

$$
\begin{aligned}
E_1 &= a \oplus' \left(\begin{pmatrix} r \\ s \end{pmatrix} \otimes \left(a' \oplus' \left(\begin{pmatrix} r' \\ s' \end{pmatrix} \otimes \begin{pmatrix} x_r \\ x_s \end{pmatrix} \right) \right) \right) \\
E_2 &= A \oplus' \left(\begin{pmatrix} R \\ S \end{pmatrix} \otimes \begin{pmatrix} x_r \\ x_s \end{pmatrix} \right)
\end{aligned}
$$

Due to space limitations, we only provide the results of the calculation.

$$
\begin{aligned}
E_1 &= \begin{pmatrix} ((a+r+a'+r')+x_r) \uparrow ((a+r) \uparrow 0) \\ (((a+r+a'+r') \uparrow (a'+r')) + x_r) \uparrow ((a+r) \uparrow s \uparrow s') \uparrow x_s \end{pmatrix} \\
E_2 &= \begin{pmatrix} (A+R+x_r) \uparrow 0 \\ (A+R+x_r) \uparrow S \uparrow x_s \end{pmatrix}
\end{aligned}
$$

Operator $\otimes$ is not extended-distributive since there are two conflicts in the calculation results above. The first is that E_2 includes two $(A + R)$'s but the corresponding parts in E_1 have difference definitions. The other is that E_2 includes constant value 0 but the corresponding part in E_1 is not a constant. To resolve these conflicts, we generalize the definition of $\oplus'$ by assigning two variables a and b to the two occurrences of a respectively, and variable c for constant 0. The definitions for the generalized operator $\oplus''$, its unit $\iota_{\oplus''}$, and the function which converts the left argument of $\oplus'$ to that of $\oplus''$ are given as follows.

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \oplus'' \begin{pmatrix} r \\ s \end{pmatrix} = \begin{pmatrix} (a+r) \uparrow c \\ (b+r) \uparrow s \end{pmatrix}, \quad \iota_{\oplus''} = \begin{pmatrix} 0 \\ -\infty \\ -\infty \end{pmatrix}, \quad conv\ a = \begin{pmatrix} a \\ a \\ 0 \end{pmatrix}$$

With operator $\oplus''$ and function *conv*, we can rewrite *mcs2′* as follows.

$$mcs2'\ (RNode\ b\ [t_1, t_2, \ldots, t_n]) = conv\ b \oplus''\ (mcs2'\ t_1 \otimes mcs2'\ t_2 \otimes \cdots \otimes mcs2'\ t_n)$$

We again validate whether $\otimes$ is extended-distributive over the newly defined $\oplus''$ by simplifying the two expressions and finding the matches between them.

$$E_1 = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \oplus'' \left(\begin{pmatrix} r_1 \\ s_1 \end{pmatrix} \otimes \left(\begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} \oplus'' \left(\begin{pmatrix} r_2 \\ s_2 \end{pmatrix} \otimes \begin{pmatrix} x_r \\ x_s \end{pmatrix} \right) \right) \right)$$

$$E_2 = \begin{pmatrix} A \\ B \\ C \end{pmatrix} \oplus'' \left(\begin{pmatrix} R \\ S \end{pmatrix} \otimes \begin{pmatrix} x_r \\ x_s \end{pmatrix} \right)$$

Now again, we only show the results of the calculation.

$$E_1 = \begin{pmatrix} ((a_1 + r_1 + a_2 + r_2) + x_r) \uparrow ((a_1 + r_1 + c_2) \uparrow c_1) \\ (((b_1 + r_1 + a_2 + r_2) \uparrow (b_2 + r_2)) + x_r) \uparrow ((b_1 + r_1 + c_2) \uparrow s_1 \uparrow s_2) \uparrow x_s \end{pmatrix}$$

$$E_2 = \begin{pmatrix} (A + R + x_r) \uparrow C \\ (B + R + x_r) \uparrow S \uparrow x_s \end{pmatrix}$$

From these results, we obtain the following correspondences.

$$\begin{aligned} A + R &= a_1 + r_1 + a_2 + r_2 \\ B + R &= (b_1 + r_1 + a_2 + r_2) \uparrow (b_2 + r_2) \\ C &= (a_1 + r_1 + c_2) \uparrow c_1 \\ S &= (b_1 + r_1 + c_2) \uparrow s_1 \uparrow s_2 \end{aligned}$$

We can obtain a solution for the correspondences above, e.g. by fixing R as 0. We then derive the following characteristic functions from the correspondences.

$$p_1 \left(\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix}, \begin{pmatrix} r_1 \\ s_1 \end{pmatrix}, \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix}, \begin{pmatrix} r_2 \\ s_2 \end{pmatrix} \right) = \begin{pmatrix} a_1 + r_1 + a_2 + r_2 \\ (b_1 + r_1 + a_2 + r_2) \uparrow (b_2 + r_2) \\ (a_1 + r_1 + c_2) \uparrow c_1 \end{pmatrix}$$

$$p_2 \left(\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix}, \begin{pmatrix} r_1 \\ s_1 \end{pmatrix}, \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix}, \begin{pmatrix} r_2 \\ s_2 \end{pmatrix} \right) = \begin{pmatrix} 0 \\ (b_1 + r_1 + c_2) \uparrow s_1 \uparrow s_2 \end{pmatrix}$$

6.4. Derive parallel program

Since we have proved the extended-distributivity of $\otimes$ over $\oplus''$ and derived the characteristic functions p_1 and p_2 in the previous step, we are ready to derive a parallel algorithm based on Corollary 1. Simply applying the operators and functions for Corollary 1, we obtain the following parallel algorithm. The two contracting operations, ϕ_L and ϕ_R, have the same definition, namely ϕ.

$$\psi_1\ (\mathit{OrgLeaf}\ a) = \begin{pmatrix} a \uparrow 0 \\ a \uparrow 0 \end{pmatrix} \qquad \psi_2\ (\mathit{OrgNode}\ b) = \left(\begin{pmatrix} b \\ b \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -\infty \end{pmatrix} \right)$$

$$\psi_1\ \mathit{OrgNil} = \begin{pmatrix} 0 \\ -\infty \end{pmatrix} \qquad \psi_2\ \mathit{Dummy} = \left(\begin{pmatrix} 0 \\ -\infty \\ -\infty \end{pmatrix}, \begin{pmatrix} 0 \\ -\infty \end{pmatrix} \right)$$

$$\phi\ \left(\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix}, \begin{pmatrix} r_1 \\ s_1 \end{pmatrix} \right)\ \begin{pmatrix} x_r \\ x_s \end{pmatrix}\ \left(\begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix}, \begin{pmatrix} r_2 \\ s_2 \end{pmatrix} \right)$$
$$= \left(\begin{pmatrix} a_1 + r_1 + x_r + a_2 + r_2 \\ (b_1 + r_1 + x_r + a_2 + r_2) \uparrow (b_2 + r_2) \\ (a_1 + r_1 + x_r + c_2) \uparrow c_1 \end{pmatrix}, \begin{pmatrix} 0 \\ (b_1 + r_1 + x_r + c_2) \uparrow s_1 \uparrow x_s \uparrow s_2 \end{pmatrix} \right)$$

$$G \left[\left(\begin{pmatrix} a \\ b \\ c \end{pmatrix}, \begin{pmatrix} r \\ s \end{pmatrix} \right) \right] = \lambda \begin{pmatrix} x_r \\ x_s \end{pmatrix} \begin{pmatrix} y_r \\ y_s \end{pmatrix} . \begin{pmatrix} a \\ b \\ c \end{pmatrix} \oplus'' \left(\begin{pmatrix} r \\ s \end{pmatrix} \otimes \begin{pmatrix} x_r \\ x_s \end{pmatrix} \otimes \begin{pmatrix} y_r \\ y_s \end{pmatrix} \right)$$

Observing the definitions of ψ_2 and ϕ above, we can find that the first value of the second tuple, i.e. r, is always 0. It follows that we can remove variable r from the definition after substituting 0 for every occurrence of r, r_1, and r_2. With this optimization, we successfully derive an efficient parallel program as shown in Fig. 3.

It is known that we need four values in the parallel program for the maximum segment sum problem on lists [3,21]. The derived parallel program is reasonably efficient, since it also uses four values despite its applicability to general trees.

7. Conclusion

We developed a new methodology for systematically deriving efficient parallel programs on trees with arbitrary shapes. Our methodology consists of three key ideas: a new formalization of conditions for the shunt contraction (Theorem 1), the extended-distributive property which generalizes associativity and distributivity, and the parallelization of a class of reductions on rose trees (Theorem 2). The formalization of conditions for the shunt contraction enables us to build a parallel program based on the tree contraction approach in a more constructive way. The extended-distributive property is so powerful that we can uniformly deal with associative operators, distributive operators, and other operators. Furthermore, we can find an extended-distributive operator systematically by generalizing the definition and examining matching. The definition of parallelizable reduction captures many

$$\psi_1\ (\mathit{OrgLeaf}\ a) = \begin{pmatrix} a \uparrow 0 \\ a \uparrow 0 \end{pmatrix} \qquad \psi_2\ (\mathit{OrgNode}\ b) = \left(\begin{pmatrix} b \\ b \\ 0 \end{pmatrix}, -\infty \right)$$

$$\psi_1\ \mathit{OrgNil} = \begin{pmatrix} 0 \\ -\infty \end{pmatrix} \qquad \psi_2\ \mathit{Dummy} = \left(\begin{pmatrix} 0 \\ -\infty \\ -\infty \end{pmatrix}, -\infty \right)$$

$$\phi \left(\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix}, s_1 \right) \begin{pmatrix} x_r \\ x_s \end{pmatrix} \left(\begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix}, s_2 \right)$$

$$= \left(\begin{pmatrix} a_1 + x_r + a_2 \\ (b_1 + x_r + a_2) \uparrow b_2 \\ (a_1 + x_r + c_2) \uparrow c_1 \end{pmatrix}, (b_1 + x_r + c_2) \uparrow s_1 \uparrow x_s \uparrow s_2 \right)$$

$$G \left[\left(\begin{pmatrix} a \\ b \\ c \end{pmatrix}, s \right) \right] = \lambda \begin{pmatrix} x_r \\ x_s \end{pmatrix} \begin{pmatrix} y_r \\ y_s \end{pmatrix} . \begin{pmatrix} a \\ b \\ c \end{pmatrix} \oplus'' \left(\begin{pmatrix} 0 \\ s \end{pmatrix} \otimes \begin{pmatrix} x_r \\ x_s \end{pmatrix} \otimes \begin{pmatrix} y_r \\ y_s \end{pmatrix} \right)$$

Fig. 3. A parallel algorithm for the maximum connected-set sum problem

tree reductions such as manipulations of structured documents discussed in [16].

The power of our method was demonstrated in the derivation of a parallel program for the maximum connected-sum problem. It is this problem that first motivated us to develop the methodology, since we could not derive a parallel program for the problem using the techniques that have been proposed so far: operators $\oplus$ and $\otimes$ do not satisfy the distributive law although $+$ and $\uparrow$ do, and the definition is not simple enough to enable us to parallelize it instinctively. In Section 6, we discussed how we systematically derived a parallel program, which is reasonably efficient. To the best of our knowledge, this is the first derivation of a parallel program for the maximum connected-sum problem on trees with arbitrary shapes.

We are currently working on generalizing this methodology to deal with recursive datatypes more efficiently. In addition, we are working on applying the extended-distributive property to other situations: for example, fusing of sequential calls of skeletons on lists and deriving general skeletons for nested lists.

References

[1] M. Cole, *Algorithmic skeletons : a structured approach to the management of parallel computation*, Research Monographs in Parallel and Distributed Computing (Pitman, London, 1989).

[2] F. Rabhi and S. Gorlatch, *Patterns and Skeletons for Parallel and Distributed Computing* (Springer-Verlag New York Inc., 2002).

[3] M. Cole, Parallel programming, list homomorphisms and the maximum segment sum problems, Report CSR-25-93, Department of Computing Science, The University of Edinburgh (1993).

[4] S. Gorlatch, Systematic efficient parallelization of scan and other list homomorphisms, in *Proc. Annual European Conference on Parallel Processing (Euro-Par '96), LNCS*

1124 (Springer-Verlag, 1996) 401–408.
[5] Z. Hu, H. Iwasaki, and M. Takeichi, Formal derivation of efficient parallel programs by construction of list homomorphisms, *ACM Trans. on Programming Languages and Systems*, **19**(3) (1997) 444–461.
[6] D. B. Skillicorn, The bird-meertens formalism as a parallel model, in *Software for Parallel Computation*, eds. J. S. Kowalik and L. Grandinetti, **106** of *NATO ASI Series F* (Springer-Verlag, 1993) 120–133.
[7] J. Ahn and T. Han, An analytical method for parallelization of recursive functions, *Parallel Process. Lett.*, **10**(1) (2000) 87–98.
[8] K. Abrahamson, N. Dadoun, D. G. Kirkpatrik, and T. Przytycka, A simple parallel tree contraction algorithm, *J. Algorithms*, **10**(2) (1989) 287–302.
[9] G. L. Miller and J. H. Reif, Parallel tree contraction and its application, in *Proc. 26th Annual Symposium on Foundations of Computer Science* (IEEE Computer Sociery Press, 1985) 478–489.
[10] G. L. Miller and J. H. Reif, Parallel tree contraction part 2: Further applications, *SIAM J. Comput.*, **20**(6) (1991) 1128–1147.
[11] J. H. Reif and S. R. Tate, Dynamic parallel tree contraction, in *Proc. the Symposium on Parallel Algorithms and Architecture* (1994) 114–121.
[12] J. Gibbons, W. Cai, and D. B. Skillicorn, Efficient parallel algorithms for tree accumulations, *Sci. Comput. Program.*, **23**(1) (1994) 1–18.
[13] D. B. Skillicorn, *Foundations of Parallel Programming* (Cambridge University Press, 1994).
[14] D. B. Skillicorn, Parallel implementation of tree skeletons, *J. Parallel Distr. Com.*, **39**(2) (1996) 115–125.
[15] D. B. Skillicorn, A parallel tree difference algorithm, *Inform. Process. Lett.*, **60**(5) (1996) 231–235.
[16] D. B. Skillicorn, Structured parallel computation in structured documents, *J. Univers. Comput. Sci.*, **3**(1) (1997) 42–68.
[17] H. Deldari, J. R. Davy, and P. M. Dew, Parallel CSG, skeletons and performance modelling, in *Proc. the Second Annual CSI Computer Conference (CSICC'96)* (1996) 115–122.
[18] K. Matsuzaki, Z. Hu, and M. Takeichi, Parallelization with tree skeletons, in *Proc. Annual European Conference on Parallel Processing (Euro-Par 2003), LNCS 2790* (Springer-Verlag, 2003) 789–798. .
[19] Z. Hu, M. Takeichi, and H. Iwasaki, Towards polytypic parallel programming, Technical Report METR 98-09, University of Tokyo (1998).
[20] J. Bentley, Column7: Algorithm design techniques, in *Programming Pearls* (Addison-Wesley, 1986) 69–80.
[21] Z. Hu, H. Iwasaki, and M. Takeichi, Construction of list homomorphisms by tupling and fusion, in *Proc. 21st International Symposium on Mathematical Foundation of Computer Science, LNCS 1113* (Springer-Verlag, 1996) 407–418.
[22] Z. Hu, H. Iwasaki, and M. Takeichi, Formal derivation of parallel program for 2-dimensional maximum segment sum problem, in *Proc. Annual European Conference on Parallel Processing (Euro-Par '96), LNCS 1123* (Springer-Verlag, 1996) 553–562.
[23] W.N. Chin, A. Takano, and Z. Hu, Parallelization via context preservation, in *Proc. IEEE Computer Society International Conference on Computer Languages (ICCL'98)* (1998) 153–162.