On the Parallel Implementation of a Generalized Broadcast

Christoph Wedler and Christian Lengauer

Fakultät für Mathematik und Informatik,
Universität Passau, 94030 Passau, Germany
Phone: +49 – 851 – 509–3071, *Fax:* +49 – 851 – 509–3092
{wedler,lengauer}@fmi.uni-passau.de
http://www.uni-passau.de/~lengauer/

Abstract. We prove the correctness of optimized parallel implementations of a generalized broadcast, in which a value b is distributed to a sequence of processors, indexed from 0 upwards, such that processor i receives $g^i\, b$ (i.e., some function g applied i times to b). Its straight-forward implementation is of linear time complexity in the number of processors. This type of broadcast occurs when combining scans with an ordinary broadcast. The optimized parallel implementations we describe is based on an odd-even tree and has logarithmic time complexity.

Keywords: broadcast, scan, parallelization, proofs.

1 Introduction

We are working on a tabular classification of linearly recursive functions and their parallel implementations. The idea is to take a general parallel linear recursion schema and perform an exhaustive study of its special cases. In this paper we deal with the class "Broadcast" which is one of the four interesting patterns Sequential, Broadcast, Reduction and Scan described in [WL97].

This class includes not only a simple broadcast function but also functional compositions of a simple broadcast with scans (generalized prefix sums). The scan is a very general and powerful computation pattern for data-parallel programming [Ble90]. The parallel scan with an associative operator has been known for some time to be of logarithmic time complexity [Lei92], although there is a linear dependence between all intermediate sums. The same result applies to functions in the class Broadcast, even if implemented naïvely as the sequential composition of a parallel broadcast and a parallel scan.

We improve further on this result by merging the two (or three, depending on the subclass) parts of the sequential composition. All optimized implementations use a binary tree which could mapped easily onto hypercubes. Of course, this optimization does not result in an implementation of better than logarithmic time complexity, but the constant factor is reduced a lot, e.g., assuming a two-phase tree implementation of a scan (up- and down-sweep) [O'D94], a functional composition of a simple broadcast with two scans would require five tree traversals, whereas our implementation of a special broadcast which is functionally equivalent to this composition requires only one tree traversal.

The algorithms presented here have already been published [WL97]. The contribution of this paper is their correctness proofs. We present a general proof schema for the data structure involved (here an odd-even tree). Its use structures the proofs and reduces their size by a huge factor compared to the naïve use of equational reasoning [Mba98]. The reason is that, in the general proof schema, one proof step covers the difficult aspects of many steps of a naïve proof: e.g., the introduction of a *tree invariant* helps us to avoid unnecessary induction proofs. See Section 2 for details.

2 Broadcast

The standard broadcast is a copy: it spreads some datum b across a set of processors (or nodes in a distribution graph). Our broadcast is slightly more general: we assume the nodes to be ordered and some function g to be applied repeatedly when the datum is propagated from one node to the next; with the identity for g, this is a copy. This type of broadcast occurs when applying scan (maybe several times) on the result of a copy. The scan is an operation which has extensive uses in data-parallel programming [Ble90]. Its most typical application yields the sum of a list of values, with intermediate results.

Because of the repeated application of g, the straight-forward implementation is of linear time complexity in the number of processors. Under specific conditions, parallel implementations with logarithmic time complexity can be used. We prove here the correctness of these implementation, in which the processors are the leaves of an odd-even tree. In Figure 1, the odd-even tree is depicted by solid arrows.

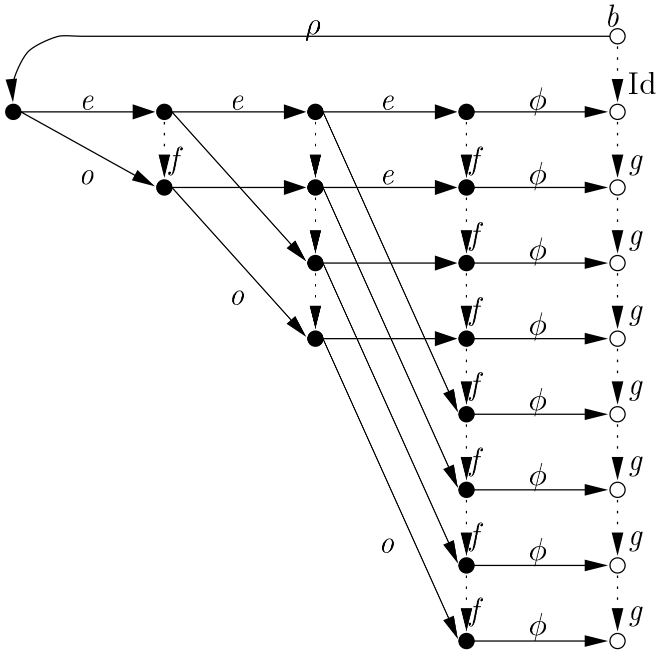

Fig. 1. Odd-even tree for implementation of Broadcast

Broadcasting a value b is defined as repeatedly applying function g:

$$broadcast_n\; g\; b = [b, g\; b, \dots, g^{n-1}\; b]$$

The computation on the tree proceeds as follows. Datum b is supplied to function ρ to compute the root value. At each non-leaf node, function e computes the value of the "even" successor, and function o computes the value of the "odd" successor. Applying function ϕ to each leaf value returns the value of the broadcast. In most cases, ϕ just discards some "accumulator" values which are used during the computation, function ρ has to provide the right initialization for the accumulator. We illustrate the idea behind some accumulators in the proofs below.

Note that for each instance of a broadcast, i.e., for each g, the implementer has to provide the functions ρ, ϕ, e and o. We have done this for three patterns of a broadcast in the subsections of Section 3.

Our statement of correctness says that the sequence of calls to g yields the same result as the tree of calls to its left. In order to be able to conduct an inductive proof, we add dotted "virtual" edges between neighboring nodes at each level and label them with function symbol f. In a sense, function f simulates g in the tree while also taking the accumulator into account. However, just like g, f need not be implemented: we shall show that the same value it would compute is already computed by functions e and o along the path to the node.

In the correctness proof, we have two obligations:

– We must show that function f simulates function g. This is expressed by Equations (1) and (2) below.
– We must show that the diagram composed of any two different paths between two nodes of the tree commutes, i.e., that the two corresponding compositions of the edge functions e, o and f yield the same value. This is expressed by Equations (3), (4) and (5).

However, these obligations are necessary only to what we call feasible node values.

Definition 1 (Feasible node value). *A* feasible node value *is a value which is derived from any input value b by any path starting with ρ and containing only solid edges in Figure 1. We denote the set of function compositions representing these paths by $\Psi^*_{\{e,o\}}$, where, for any set F of functions, Ψ^*_F is defined recursively by:*

$$\Psi^*_F = \{\mathrm{Id}\} \cup \{f \circ \psi \mid f \in F, \psi \in \Psi^*_F\}$$

In some proofs (those of Equations (2), (3) and (4)), we characterize the set of feasible node values by an invariant.

Definition 2 (Invariant of odd-even tree). *A predicate Φ is called a* tree invariant *if it is true for all feasible node values of the tree:*

$$\forall \psi \in \Psi^*_{\{e,o\}} \,.\, \forall b \,.\, \Phi\,((\psi \circ \rho)\; b)$$

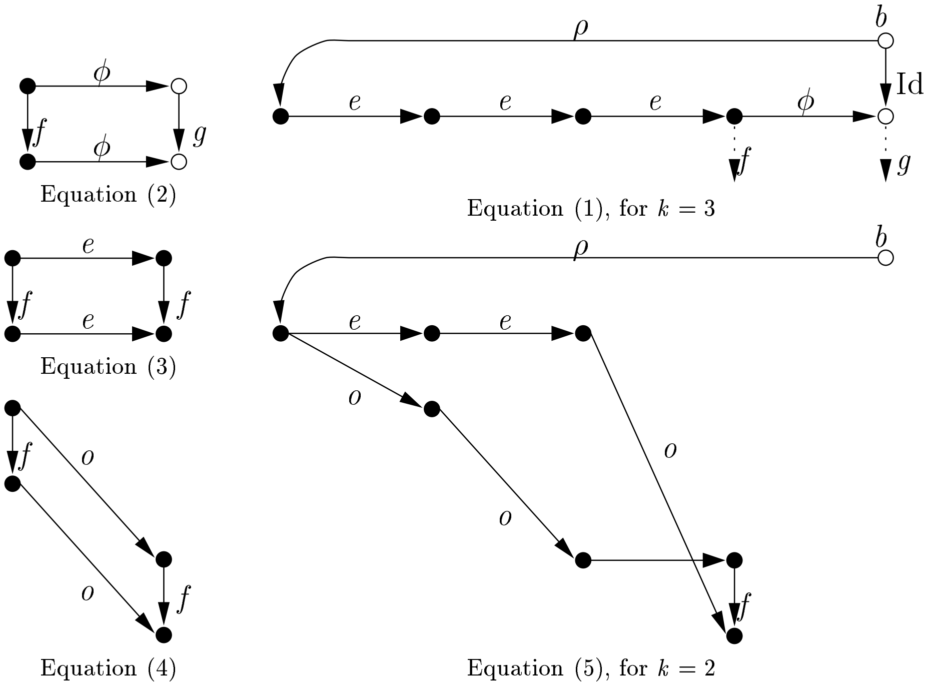

Fig. 2. Equations for correct implementations using an odd-even tree

Let us now state Equations (1)–(5), which we have been alluding to. In Figure 2, we illustrate them by repeating the relevant parts of the odd-even tree in Figure 1. Solid arrows represent functions appearing in the equations, dotted arrows those only appearing in the explanation.

Function f starts with the right value, i.e., the value at the uppermost leaf of the odd-even tree corresponds to the input value b which is the first input value of function g.

$$\forall k \geq 0 \,.\, \phi \circ e^k \circ \rho = \mathrm{Id} \tag{1}$$

Function f simulates function g. The values at the leaves correspond to the output values of the broadcast:

$$\forall x \,.\, \Phi\, x \Rightarrow (\phi \circ f)\, x = (g \circ \phi)\, x \tag{2}$$

At any node in the tree, the functional compositions of f with e and f with o are commutative:

$$\forall x \,.\, \Phi\, x \Rightarrow (f \circ e)\, x = (e \circ f)\, x \tag{3}$$

$$\forall x \,.\, \Phi\, x \Rightarrow (f \circ o)\, x = (o \circ f)\, x \tag{4}$$

The lowest node of the upper half of the nodes at one level is the direct predecessor of the highest node of the lower half of the nodes at the same level:

$$\forall k \geq 0 \,.\, f \circ e \circ o^k \circ \rho = o \circ e^k \circ \rho \tag{5}$$

The use of the tree invariant immediately becomes clear when looking at Equations (2), (3) and (4): only the proof of the invariant requires structural induction, the proofs of these three equations are simple.

3 The Proofs

In the past, we have studied three different broadcast patterns: CS-Broadcast, CSS-Broadcast and CSS2-Broadcast [WL97] which occur when combining one or two scans with an ordinary broadcast.

We relegate lemmas and proofs which are straight-forward to the appendix; some are omitted altogether in the interest of brevity. Throughout the paper, the operators $\oplus$ and $\otimes$ are associative; we do not state these properties explicitly in the proofs, nor do we refer to the defining equation of the functional composition $(f \circ g)\, x = f\,(g\, x)$.

3.1 CS-Broadcast

First, let us explain where this kind of broadcast pattern occurs. Copying some value, say, 2 (in general, b_0) to 8 places (in general, n) returns the list $[2,2,2,2,2,2,2,2]$. If we compute the prefix product over this list, starting with 1 (in general, $scan\,(\oplus)\,b_1$), we get the list $[1,2,4,8,16,32,64,128]$. This combination of a copy and a scan turns out to be part of the result of a single CS-Broadcast [WL97], which is the list $[(2,1),(2,2),(2,4),(2,8),(2,16),(2,32),(2,64),(2,128)]$.

Functions which can be expressed as a CS-Broadcast can be instantiated with

$$broadcast_n\; g\,(b_0,b_1) \qquad \text{where} \qquad g\,(b_0,b_1)=(b_0,b_1\oplus b_0), \quad \oplus \text{ is associative}$$

The computation on the odd-even tree, for $n=8$, $b_0=2$, $b_1=1$ and $\oplus = *$, is given by Figure 3.

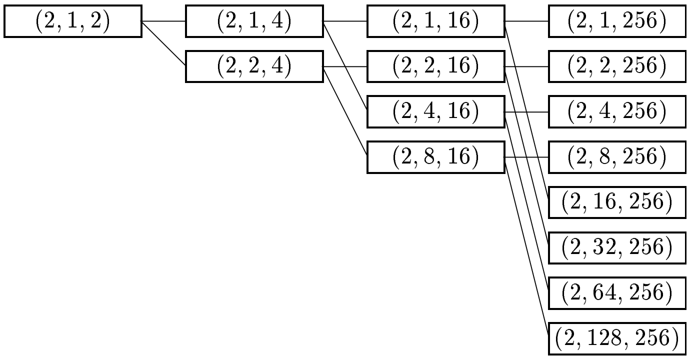

Fig. 3. Example computation in the odd-even tree for CS-Broadcast

This computation is quite similar to the efficient evaluation of powers [Knu80, Sect. 4.6.3]. The accumulator r stores $\bigoplus^{2^k} b_0$, where k is the level in the tree ($k=0$: root, $k=\log_2 n$: leaves). Equation (8) states that the value of this accumulator is identical for all nodes at one level, Equations (9) and (10) state that the values are squared with each transition to a new level.

The accumulator is initialized with ρ and discarded with ϕ:

$$\rho\,(b_0,b_1)=(b_0,b_1,b_0) \tag{6}$$
$$\phi\,(p,q,r)=(p,q) \tag{7}$$

Function g is simulated by function f:

$$f\,(p,q,r)=(p,q\oplus p,r) \tag{8}$$

The functions which compute the values in the nodes of the next level are:

$$e\,(p,q,r)=(p,q,r\oplus r) \tag{9}$$
$$o\,(p,q,r)=(p,q\oplus r,r\oplus r) \tag{10}$$

In the proof of Lemma 9 in the appendix, we need to know that, for each feasible node value (p,q,r), equation $p\oplus r = r\oplus p$ holds. At first sight, we might think that this implementation of a CS-Broadcast can only be used if $\oplus$ is commutative. But the invariant Φ below states that it applies even if $\oplus$ is not commutative.

Lemma 1 (Invariant CS-Broadcast). *The following predicate holds for the values at each node in the tree for the implementation of CS-Broadcast:*

$$\Phi\,(p,q,r) \equiv p\oplus r = r\oplus p \tag{11}$$

Proof. by induction on the path from the root to each node; see the appendix.

Informally, the correctness of the invariant can be justified by the fact that r is the accumulator for $\bigoplus^i p$, where $i=2^k$ and $p=b_0$. It is clear that $p\oplus(\bigoplus^i p)=(\bigoplus^i p)\oplus p$.

The proofs of Equations (1), (2) and (3) are simple; we omit them here. The proof of Equation (4) is as simple as the proof of Equation (3) but uses invariant Φ. We prove Equation (4) as Lemma 9 in the appendix.

Only the proof of Equation (5) in Lemma 2 is not straight-forward. It is based on a generalization, which is stated in Lemma 3.

Lemma 2. *Equation* (5) *is valid.*

Proof.

$$
\begin{aligned}
& (f \circ e \circ o^k)\,(b_0, b_1, b_0) \\
= \quad & \{\, \textit{Lemma 3, since } 1_\oplus \oplus 1_\oplus \oplus b_0 = 1_\oplus \oplus b_0 \oplus 1_\oplus \,\} \\
& (o \circ e^k)\,(b_0, b_1, b_0)
\end{aligned}
$$

Lemma 3. *Generalization of Lemma 2* (Explanation follows).

$$
\begin{aligned}
& \forall k \geq 0 \,.\, \forall p, q, \delta \,.\, \delta \oplus \delta \oplus p = \delta \oplus p \oplus \delta \Rightarrow \\
& \quad (f \circ e \circ o^k)\,(p, q \oplus \delta, \delta \oplus p) = (o \circ e^k)\,(p, q, \delta \oplus p)
\end{aligned}
\tag{12}
$$

Proof. Induction over k, see the appendix. Lemma 10 proves the premise for the induction hypothesis.

Lemma 3 looks more complicated than it is. The second value q of the triple in the nodes (see Figure 3) is of interest here:

- At the top node of each level k ($\psi = e^k$), q is unchanged.
- At the bottom node ($\psi = o^k$), q can be computed by applying function f $2^k - 1$ times, which adds the difference $\delta = \bigoplus^{2^k-1} p$ to q.
- The accumulator stores value $r = \bigoplus^{2^k} p$, which can be expressed as $\delta \oplus p$.
- Starting from the last node, function e does not change q, function f adds p to it. Thus, we have $(q \oplus \delta) \oplus p$.
- Starting from the first node, function o adds r to q. Thus, we have $q \oplus (\delta \oplus p)$.

3.2 CSS-Broadcast

In a CS-Broadcast, we scan over the result of a copy. In a CSS-Broadcast, we scan not once but twice.

Again, let us take a specific example. Copying some value, say, 1 (in general, b_0) to 8 places (in general, n) returns the list $[1, 1, 1, 1, 1, 1, 1, 1]$. If we compute the prefix sum over this list, starting with 3 (in general, $\mathit{scan}\,(\oplus)\,b_1$), we get the list $[3, 4, 5, 6, 7, 8, 9, 10]$. If we compute one more prefix sum over the result list, starting with 0 (in general, $\mathit{scan}\,(\oplus)\,b_2$), we get the list [0,3,7,12,18,25,33,42]. This combination of a copy and two scans with the same operator turns out to be part of the result of a single CSS-Broadcast [WL97], which is the list $[(1,3,0), (1,4,3), (1,5,7), (1,6,12), (1,7,18), (1,8,25), (1,9,33), (1,10,42)]$.

Functions which can be expressed as a CSS-Broadcast can be instantiated with

$$
\begin{aligned}
& \mathit{broadcast}_n\; g\,(b_0, b_1, b_2) \qquad \text{where} \qquad g\,(b_0, b_1, b_2) = (b_0, b_1 \oplus b_0, b_2 \oplus b_1), \\
& \quad \oplus \text{ is associative and commutative}
\end{aligned}
$$

The computation on the odd-even tree, for $n = 8$, $b_0 = 1$, $b_1 = 3$, $b_2 = 0$ and $\oplus = +$, is given by Figure 4.

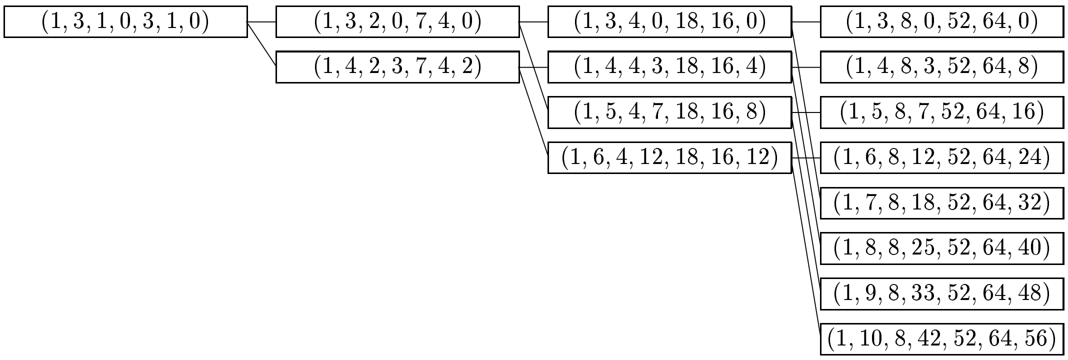

Fig. 4. Example computation in the odd-even for CSS-Broadcast

Since a part of the result of a CSS-Broadcast is the result of a CS-Broadcast (the first two values in the triples), the computation for p, q and r remains unchanged. But, just like r was needed as an accumulator in the computation of the CS-Broadcast, we need some accumulators for the CSS-Broadcast, too: t, u and v.

For illustration, let us explain how we arrived at the form of accumulator u in our functions e and o below, but we omit the intuition behind t and v.

Take the example in Figure 4. We would like to compute the intermediate results of the sum $\sum_{i=0}^{7} 3 + i$, where i is really the sum $\sum_{j=0}^{i-1} 1$. The sum can be simplified to $x_0 * n(n-1) + x_1 * n + x_2$ with appropriate constants x_i (which may depend on b_0, b_1 and b_2, but not on n).

In general, the sums we consider are of the form $b_2 \oplus \bigoplus_{i=0}^{n-1} b_1 \oplus \bigoplus_{j=0}^{i-1} b_0$. If $\oplus$ is commutative, this sum can be simplified to $(\bigoplus^{n(n-1)} x_0) \oplus (\bigoplus^{n} x_1) \oplus x_2$ with appropriate constants x_i (which may depend on b_0, b_1 and b_2). With a CSS-Broadcast, we can compute the sum (and its intermediate results) with the sole operator $\oplus$ in time $\mathcal{O}(\log_2 n)$.

In Section 3.1, we have $n = 2^k$. There, we used that, with any associative $\oplus$, we can compute $\bigoplus^{n} x_1$ in k steps of applying $r \mapsto r \oplus r$, starting with x_1. The difficult part of our sum here is $\bigoplus^{n(n-1)} x_0$, which is in the order of $\bigoplus^{n^2} x_0$. With $n^2 = 4^k$, we can compute $\bigoplus^{n^2} x_0$ in k steps of applying $u \mapsto u \oplus u \oplus u \oplus u$, starting with x_0.

All accumulators are initialized with ρ and discarded with ϕ:

$$\rho\,(b_0, b_1, b_2) = (b_0, b_1, b_0, b_2, b_1, b_0, 1_\oplus) \tag{13}$$

$$\phi\,(p, q, r, s, t, u, v) = (p, q, s) \tag{14}$$

Function g is simulated by function f:

$$f\,(p, q, r, s, t, u, v) = (p, q \oplus p, r, s \oplus q, t, u, v \oplus r) \tag{15}$$

The functions which compute the values in the nodes of the next level are:

$$e\,(p, q, r, s, t, u, v) = (p, q, r \oplus r, s, t \oplus t \oplus u, u \oplus u \oplus u \oplus u, v \oplus v) \tag{16}$$

$$\begin{aligned} o\,(p, q, r, s, t, u, v) = (&p, q \oplus r, r \oplus r, s \oplus t \oplus v, t \oplus t \oplus u, \\ &u \oplus u \oplus u \oplus u, u \oplus u \oplus v \oplus v) \end{aligned} \tag{17}$$

The proofs of Equations (1)–(4) are simple; we omit them here.

Only the proof of Equation (5) in Lemma 4 is not straight-forward. Like the corresponding proof for the CS-Broadcast, it is based on a generalization, which is stated in Lemma 5.

Lemma 4. *Equation* (5) *is valid.*

Proof.

$$\begin{aligned} & (f \circ e \circ o^k)\,(b_0, b_1, b_0, b_2, b_1, b_0, 1_\oplus) \\ = \quad & \{\, \textit{Lemma 5} \,\} \\ & (o \circ e^k)\,(b_0, b_1, b_0, b_2, b_1, b_0, 1_\oplus) \end{aligned}$$

Lemma 5. *Generalization of Lemma 4.*

$$\begin{aligned} & \forall k \geq 0 \,.\, \forall p, q, s, \delta, \lambda, \mu \,. \\ & \quad (f \circ e \circ o^k)\,(p, q \oplus \delta, \delta \oplus p, s \oplus \lambda, q \oplus \delta \oplus \lambda, p \oplus \delta \oplus \mu, \mu) \\ & \quad = (o \circ e^k)\,(p, q, \delta \oplus p, s, q \oplus \delta \oplus \lambda, p \oplus \delta \oplus \mu, 1_\oplus) \end{aligned}$$

Proof. Induction over k, see the appendix.

The explanation for Lemma 5 is similar to the explanation for Lemma 3. Instead of one difference δ, we have three: δ, λ and μ.

3.3 CSS2-Broadcast

This type of broadcast pattern differs from the CSS-Broadcast in that the subsequent scan has another operator $\otimes$ than the operator $\oplus$ of the scan. Again, this combination can be computed with a single Broadcast, thus avoiding one scan.

Functions which can be expressed as a CSS-Broadcast can be instantiated with

$$\textit{broadcast}_n\; g\,(b_0, b_1, b_2) \qquad \text{where} \qquad g\,(b_0, b_1, b_2) = (b_0, b_1 \otimes b_0, b_2 \oplus b_1),$$

$\oplus$ and $\otimes$ are associative, $\otimes$ distributes over $\oplus$, neutral element of $\oplus$ is the same as the absorbing zero element of $\otimes$: $1_\oplus = 0_\otimes$

Again, two of the three values in the elements of the result are identical to the elements of the result of CS-Broadcast, as are some parts of the computation.

All accumulators are initialized with ρ and discarded with ϕ:

$$\rho\,(b_0, b_1, b_2) = (b_0, b_1, b_0, 1_\oplus, b_1, b_0, b_2) \tag{18}$$
$$\phi\,(p, q, r, s, t, v) = (p, q, v \oplus s) \tag{19}$$

Function g is simulated by function f:

$$f\,(p, q, r, s, t, v) = (p, q \otimes p, r, s \oplus q, t, v) \tag{20}$$

The functions which compute the values in the nodes of the next level are:

$$e\,(p, q, r, s, t, u) = (p, q, r \otimes r, s, t \oplus (t \otimes r), v) \tag{21}$$
$$o\,(p, q, r, s, t, v) = (p, q \otimes r, r \otimes r, t \oplus (s \otimes r), t \oplus (t \otimes r), v) \tag{22}$$

The invariant for feasible values during the computation of a CSS2-Broadcast is identical to the invariant in Section 2. It uses of the six node values only the two values p and q which are also used in the computation of a CS-Broadcast.

Lemma 6 (Invariant CSS2-Broadcast). *The following predicate holds for the values at each node in the tree for the implementation of CSS2-Broadcast:*

$$\Phi\,(p, q, r, s, t, v) \equiv p \otimes r = r \otimes p \tag{23}$$

Proof. *See the proof of Lemma 1.*

The proofs of Equations (1)–(3) are simple; we omit them here. The proof of Equation (4) is as simple as the proof of Equation (3) but uses invariant Φ. We prove Equation (4) as Lemma 11 in the appendix.

Only the proof of Equation (5) in Lemma 7 is not straight-forward. Like the corresponding proof for the CS-Broadcast, it is based on a generalization, which stated in Lemma 8.

Lemma 7. *Equation* (5) *is valid.*

Proof.

$$\begin{array}{ll} & (f \circ e \circ o^k)\,(b_0, b_1, b_0, 1_\oplus, b_1, b_0, b_2) \\ = & \{\,\textit{Lemma 8, since } 1_\oplus \oplus 1_\oplus \oplus b_0 = 1_\oplus \oplus b_0 \oplus 1_\oplus\,\} \\ & (o \circ e^k)\,(b_0, b_1, b_0 1_\oplus, b_1, b_0, b_2) \end{array}$$

Lemma 8. *Generalization of Lemma 7.*

$$\begin{array}{l} \forall k \geq 0\,.\,\forall p, q, v, \delta, \gamma\,.\,\delta \otimes \delta \otimes p = \delta \otimes p \otimes \delta \Rightarrow \\ \quad (f \circ e \circ o^k)\,(p, q \otimes \delta, \delta \otimes p, \gamma, \gamma \oplus (q \otimes \delta), v) \\ \quad = (o \circ e^k)\,(p, q, \delta \otimes p, 1_\oplus, \gamma \oplus (q \otimes \delta), v) \end{array}$$

Proof. *Induction over k, see the appendix.*

4 Conclusions

We have presented the correctness proofs of optimized implementations of a number of generalized broadcast patterns which arise in parallel scans of broadcast information.

There has been related work in our own research group as well as elsewhere. Transformation rules for combinations of scans with other parallel operations have been proved formally and used for program derivation [CS95,Gor97]. The performance gain and overhead of composition and decomposition rules have been assessed analytically for the hypercube and compared with the estimates for some other parallel models [GL98]. This work makes use of the theory of list homomorphisms [Bir89,Ski94], which does not apply in the case of a generalized broadcast since its input is not a list.

We have expressed our implementations as computations on an odd-even tree. The edges represent functions which are applied to the value on the source node to compute the value of the target node. We have enriched this data structure by "virtual" edges representing functions which would have been applied if the broadcasts had been implemented naïvely following the specification.

A proof then consists of showing that the function applications along different paths in the graph to the same node return the same value. To that purpose, we have stated a set of equations which had to be proven correct.

Of course, our approach of specifying different paths in the enriched data structure by equations applies more generally. Consequently, proof obligations similar to ours for the odd-even tree can be expressed. Interestingly enough, equations which have a form similar to ours are also used to find unique node labels for pointer-based data structures with different paths to the same node [LA98].

Acknowledgements

This work has profited from discussions with D. K. Arvind and Tim Lewis, undertaken in an ARC exchange project funded by the DAAD. Thanks to S. Gorlatch for comments on a draft version of this paper.

References

[Bir89] Richard S. Bird. Lectures on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computing Science (Internat. Summer School 1988, Marktoberdorf, Germany)*, volume 55 of *NATO ASI Series F*, pages 150–216. Springer-Verlag, 1989.

[Ble90] Guy E. Blelloch. *Vector Models for Data-Parallel Computing.* MIT Press, 1990.

[CS95] Wentong Cai and David B. Skillicorn. Calculating recurrences using the Bird-Meertens formalism. *Parallel Processing Letters*, 5(2):179–190, 1995.

[GL98] Sergei Gorlatch and Christian Lengauer. (De)composition rules for parallel scan and reduction. In *3rd Int. Working Conf. on Massively Parallel Programming Models (MPPM'97).* IEEE Computer Society Press, 1998. To appear.

[Gor97] Sergei Gorlatch. Optimizing compositions of scans and reductions in parallel program derivation. Technical Report MIP-9711, Fakultät für Mathematik und Informatik, Universität Passau, May 1997.

[Knu80] Donald E. Knuth. *The Art of Computer Programming*, volume 2: Seminumeral Algorithms. Addison-Wesley, 2nd edition, 1980.

[LA98] Tim Lewis and D. K. Arvind. Static analysis of recursive data structures. In Z. Li, P.-C. Yew, S. Chatterjee, et al., editors, *10th Int. Workshop on Languages and Compilers for Parallel Computing* (LCPC'97), volume 1366 of *Lecture Notes in Computer Science*, pages 423–426. Springer-Verlag, 1998.

[Lei92] Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1992.

[Mba98] Roger P. Mba. Gleichungsbeweise paralleler Implementierungen linear rekursiver Programme. Diplomarbeit, Fakultät für Mathematik und Informatik, Universität Passau, Mar. 1998.

[O'D94] John T. O'Donnell. A correctness proof of parallel scan. *Parallel Processing Letters*, 4(3):329–338, 1994.

[Ski94] David B. Skillicorn. *Foundations of Parallel Programming.* Cambridge University Press, 1994.

[WL97] Christoph Wedler and Christian Lengauer. Parallel implementations of combinations of broadcast, reduction and scan. In Gul Agha and Stefano Russo, editors, *2nd Int. Workshop on Software Engineering for Parallel and Distributed Systems* (PDSE'97), pages 108–119. IEEE Computer Society Press, 1997.

A Proofs

Proof (Lemma 1).

INDUCTION BASE ($\psi = \mathrm{Id}$):

$$\begin{aligned}
& p \oplus r \\
= \quad & \{\, (p, q, r) = \rho\,(b_0, b_1) = (b_0, b_1, b_0) \,\} \\
& b_0 \oplus b_0 \\
= \quad & \{\, \rho\,(b_0, b_1) = (b_0, b_1, b_0) \,\} \\
& r \oplus p
\end{aligned}$$

INDUCTION STEP ($\psi = e \circ \psi'$):

$$\begin{aligned}
& p \oplus r \\
= \quad & \{\, (p, q, r) = e\,(p', q', r') = (p', q', r' \oplus r') \text{ where } (p', q', r') = (\psi' \circ \rho)\,(b_0, b_1) \,\} \\
& p' \oplus r' \oplus r' \\
= \quad & \{\, \textsc{Induction Hyphothesis} \,\} \\
& r' \oplus p' \oplus r' \\
= \quad & \{\, \textsc{Induction Hyphothesis} \,\} \\
& r' \oplus r' \oplus p' \\
= \quad & \{\, (p, q, r) = e\,(p', q', r') = (p', q', r' \oplus r') \,\} \\
& r \oplus p
\end{aligned}$$

INDUCTION STEP ($\psi = o \circ \psi'$):

$$\begin{aligned}
& p \oplus r \\
= & \quad \{ (p, q, r) = o\,(p', q', r') = (p', q' \oplus r', r' \oplus r') \} \\
& p' \oplus r' \oplus r' \\
= & \quad \{ \text{INDUCTION HYPHOTHESIS} \} \\
& r' \oplus p' \oplus r' \\
= & \quad \{ \text{INDUCTION HYPHOTHESIS} \} \\
& r' \oplus r' \oplus p' \\
= & \quad \{ (p, q, r) = e\,(p', q', r') = (p', q' \oplus r', r' \oplus r') \} \\
& r \oplus p
\end{aligned}$$

Lemma 9. *Equation* (4) *is valid (using invariant Φ):*

Proof.

$$\begin{aligned}
& (f \circ o)\,(p, q, r) \\
= & \quad \{ \text{DEF.}\,(o),\ \text{DEF.}\,(f) \} \\
& (p, q \oplus r \oplus p, r \oplus r) \\
= & \quad \{ \text{DEF.}\,(\text{INVARIANT } \Phi) \} \\
& (p, q \oplus p \oplus r, r \oplus r) \\
= & \quad \{ \text{DEF.}\,(f),\ \text{DEF.}\,(o) \} \\
& (o \circ f)\,(p, q, r)
\end{aligned}$$

Proof (Lemma 3).

INDUCTION BASE ($k = 0$):

$$\begin{aligned}
& (f \circ e)\,(p, q \oplus \delta, \delta \oplus p) \\
= & \quad \{ \text{DEF.}\,(e),\ \text{DEF.}\,(f) \} \\
& (p, q \oplus \delta \oplus p, \delta \oplus p \oplus \delta \oplus p) \\
= & \quad \{ \text{DEF.}\,(o) \} \\
& o\,(p, q, \delta \oplus p)
\end{aligned}$$

INDUCTION STEP ($k \longrightarrow k+1$):

$$\begin{aligned}
& (f \circ e \circ o^{k+1})\,(p, q \oplus \delta, \delta \oplus p) \\
= & \quad \{ \text{DEF.}\,(o) \} \\
& (f \circ e \circ o^{k})\,(p, q \oplus \delta \oplus \delta \oplus p, \delta \oplus p \oplus \delta \oplus p) \\
= & \quad \{ \delta \oplus \delta \oplus p = \delta \oplus p \oplus \delta \} \\
& (f \circ e \circ o^{k})\,(p, q \oplus \delta \oplus p \oplus \delta, \delta \oplus p \oplus \delta \oplus p) \\
= & \quad \{ \delta' = \delta \oplus p \oplus \delta \} \\
& (f \circ e \circ o^{k})\,(p, q \oplus \delta', \delta' \oplus p) \\
= & \quad \{ \text{INDUCTION HYPHOTHESIS since Lemma 10} \} \\
& (o \circ e^{k})\,(p, q, \delta' \oplus p) \\
= & \quad \{ \delta' = \delta \oplus p \oplus \delta \} \\
& (o \circ e^{k})\,(p, q, \delta \oplus p \oplus \delta \oplus p) \\
= & \quad \{ \text{DEF.}\,(e) \} \\
& (o \circ e^{k+1})\,(p, q, \delta \oplus p)
\end{aligned}$$

Lemma 10. *The induction hypothesis in the proofs of Lemmas 3 and 8 apply.*

$$\forall \delta, p, \delta' \textbf{,}\ \delta' = \delta \oplus p \oplus \delta \ \textbf{.}\ \delta \oplus \delta \oplus p = \delta \oplus p \oplus \delta \Rightarrow \delta' \oplus \delta' \oplus p = \delta' \oplus p \oplus \delta' \tag{24}$$

Proof.

$$
\begin{aligned}
& \delta' \oplus \delta' \oplus p \\
= \quad & \{\, \delta' = \delta \oplus p \oplus \delta \,\} \\
& \delta \oplus p \oplus \delta \oplus \delta \oplus p \oplus \delta \oplus p \\
= \quad & \{\, \delta \oplus \delta \oplus p = \delta \oplus p \oplus \delta \,\} \\
& \delta \oplus p \oplus \delta \oplus p \oplus \delta \oplus \delta \oplus p \\
= \quad & \{\, \delta \oplus \delta \oplus p = \delta \oplus p \oplus \delta \,\} \\
& \delta \oplus p \oplus \delta \oplus p \oplus \delta \oplus p \oplus \delta \\
= \quad & \{\, \delta' = \delta \oplus p \oplus \delta \,\} \\
& \delta' \oplus p \oplus \delta'
\end{aligned}
$$

Proof (Lemma 5).

Induction Base ($k = 0$):

$$
\begin{aligned}
& (f \circ e)\,(p, q \oplus \delta, \delta \oplus p, s \oplus \lambda, q \oplus \delta \oplus \lambda, p \oplus \delta \oplus \mu, \mu) \\
= \quad & \{\, \textsc{Def.}\,(e),\ \textsc{Def.}\,(f) \,\} \\
& (p, q \oplus \delta \oplus p, \delta \oplus p \oplus \delta \oplus p, s \oplus \lambda \oplus q \oplus \delta, \\
& (\textstyle\bigoplus^2 q \oplus \delta \oplus \lambda) \oplus p \oplus \delta \oplus \mu, (\textstyle\bigoplus^4 p \oplus \delta \oplus \mu), \mu \oplus \mu \oplus \delta \oplus p \oplus \delta \oplus p) \\
= \quad & \{\, \oplus \text{ is commutative} \,\} \\
& (p, q \oplus \delta \oplus p, \delta \oplus p \oplus \delta \oplus p, s \oplus q \oplus \delta \oplus \lambda, \\
& (\textstyle\bigoplus^2 q \oplus \delta \oplus \lambda) \oplus p \oplus \delta \oplus \mu, (\textstyle\bigoplus^4 p \oplus \delta \oplus \mu), p \oplus \delta \oplus \mu \oplus p \oplus \delta \oplus \mu) \\
= \quad & \{\, \textsc{Def.}\,(o) \,\} \\
& o\,(p, q, \delta \oplus p, s, q \oplus \delta \oplus \lambda, p \oplus \delta \oplus \mu, 1_{\oplus})
\end{aligned}
$$

Induction Step ($k \longrightarrow k+1$):

$$
\begin{aligned}
& (f \circ e \circ o^{k+1})\,(p, q \oplus \delta, \delta \oplus p, s \oplus \lambda, q \oplus \delta \oplus \lambda, p \oplus \delta \oplus \mu, \mu) \\
= \quad & \{\, \textsc{Def.}\,(o) \,\} \\
& (f \circ e \circ o^{k})\,(p, q \oplus \delta \oplus \delta \oplus p, \delta \oplus p \oplus \delta \oplus p, s \oplus \lambda \oplus q \oplus \delta \oplus \lambda \oplus \mu, \\
& (\textstyle\bigoplus^2 q \oplus \delta \oplus \lambda) \oplus p \oplus \delta \oplus \mu, (\textstyle\bigoplus^4 p \oplus \delta \oplus \mu), p \oplus \delta \oplus \mu \oplus p \oplus \delta \oplus \mu \oplus \mu \oplus \mu) \\
= \quad & \left\{ \begin{array}{l} \oplus \text{ is commutative, } \delta' = \delta \oplus \delta \oplus p,\ \lambda' = q \oplus \delta \oplus \lambda \oplus \lambda \oplus \mu, \\ \mu' = p \oplus p \oplus \delta \oplus \delta \oplus \mu \oplus \mu \oplus \mu \oplus m \end{array} \right\} \\
& (f \circ e \circ o^{k})\,(p, q \oplus \delta', \delta' \oplus p, s \oplus \lambda', q \oplus \delta' \oplus \lambda', p \oplus \delta' \oplus \mu', \mu') \\
= \quad & \{\, \textsc{Induction Hyphothesis} \,\} \\
& (o \circ e^{k})\,(p, q, \delta' \oplus p, s, q \oplus \delta' \oplus \lambda', p \oplus \delta \oplus \mu, 1_{\oplus}) \\
= \quad & \left\{ \begin{array}{l} \oplus \text{ is commutative, } \delta' = \delta \oplus \delta \oplus p,\ \lambda' = q \oplus \delta \oplus \lambda \oplus \lambda \oplus \mu, \\ \mu' = p \oplus p \oplus \delta \oplus \delta \oplus \mu \oplus \mu \oplus \mu \oplus m \end{array} \right\} \\
& (o \circ e^{k})\,(p, q, \delta \oplus p \oplus \delta \oplus p, s, \\
& (\textstyle\bigoplus^2 q \oplus \delta \oplus \lambda) \oplus p \oplus \delta \oplus \mu, (\textstyle\bigoplus^4 p \oplus \delta \oplus \mu), 1_{\oplus}) \\
= \quad & \{\, \textsc{Def.}\,(e) \,\} \\
& (o \circ e^{k+1})\,(p, q, \delta \oplus p, s, q \oplus \delta \oplus \lambda, p \oplus \delta \oplus \mu, 1_{\oplus})
\end{aligned}
$$

Lemma 11. *Equation* (4) *is valid:*

Proof.

$$
\begin{aligned}
& (f \circ o)\,(p, q, r, s, t, v) \\
= \quad & \{\, \textsc{Def.}\,(o),\ \textsc{Def.}\,(f) \,\} \\
& (p, q \otimes r \otimes p, r \otimes r, t \oplus (s \otimes r) \oplus (q \otimes r), t \oplus (t \otimes r), v) \\
= \quad & \{\, \textsc{Def.}\,(\textsc{Invariant}\ \Phi),\ \otimes \textit{ distributes over } \oplus \,\} \\
& (p, q \otimes p \otimes r, r \otimes r, t \oplus ((s \oplus q) \otimes r), t \oplus (t \otimes r), v) \\
= \quad & \{\, \textsc{Def.}\,(f),\ \textsc{Def.}\,(o) \,\} \\
& (o \circ f)\,(p, q, r, s, t, v)
\end{aligned}
$$

Proof (Lemma 8).

Induction Base ($k = 0$):

$$
\begin{aligned}
& (f \circ e)\,(p, q \otimes \delta, \delta \otimes p, \gamma, \gamma \oplus (q \otimes \delta), v) \\
= \quad & \{\, \text{Def.}\,(e),\ \text{Def.}\,(f) \,\} \\
& (p, q \otimes \delta \otimes p, \delta \otimes p \otimes \delta \otimes p, \gamma \oplus (q \otimes \delta), \gamma \oplus (q \otimes \delta) \oplus ((\gamma \oplus (q \otimes \delta)) \otimes \delta \otimes p), v) \\
= \quad & \{\, \text{Def.}\,(o),\ 1_\oplus = 0_\otimes \,\} \\
& o\,(p, q, \delta \otimes p, 1_\oplus, \gamma \oplus (q \otimes \delta), v)
\end{aligned}
$$

Induction Step ($k \longrightarrow k+1$):

$$
\begin{aligned}
& (f \circ e \circ o^{k+1})\,(p, q \otimes \delta, \delta \otimes p, \gamma, \gamma \oplus (q \otimes \delta), v) \\
= \quad & \{\, \text{Def.}\,(o) \,\} \\
& (f \circ e \circ o^{k})\,(p, q \otimes \delta \otimes \delta \otimes p, \delta \otimes p \otimes \delta \otimes p, \gamma \oplus (q \otimes \delta) \oplus (\gamma \otimes \delta \otimes p), \\
& \gamma \oplus (q \otimes \delta) \oplus ((\gamma \oplus (q \otimes \delta)) \otimes \delta \otimes p), v) \\
= \quad & \{\, \delta \otimes \delta \otimes p = \delta \otimes p \otimes \delta,\ \otimes \text{ distributes over } \oplus \,\} \\
& (f \circ e \circ o^{k})\,(p, q \otimes \delta \otimes p \otimes \delta, \delta \otimes p \otimes \delta \otimes p, \gamma \oplus (q \otimes \delta) \oplus (\gamma \otimes \delta \otimes p), \\
& \gamma \oplus (q \otimes \delta) \oplus (\gamma \otimes \delta \otimes p) \oplus (q \otimes \delta \otimes p \otimes \delta), v) \\
= \quad & \{\, \delta' = \delta \oplus \delta \oplus p,\ \gamma' = \gamma \oplus (q \otimes \delta) \oplus (\gamma \otimes \delta \otimes p) \,\} \\
& (f \circ e \circ o^{k})\,(p, q \otimes \delta', \delta' \otimes p, \gamma', \gamma' \oplus (q \otimes \delta'), v) \\
= \quad & \{\, \text{Induction Hyphothesis since Lemma 10} \,\} \\
& (o \circ e^{k})\,(p, q, \delta' \otimes p, 1_\oplus, \gamma' \oplus (q \otimes \delta'), v) \\
= \quad & \{\, \delta' = \delta \oplus \delta \oplus p,\ \gamma' = \gamma \oplus (q \otimes \delta) \oplus (\gamma \otimes \delta \otimes p) \,\} \\
& (o \circ e^{k})\,(p, q, \delta \otimes p \otimes \delta \otimes p, 1_\oplus, \gamma \oplus (q \otimes \delta) \oplus (\gamma \otimes \delta \otimes p) \oplus (q \otimes \delta \otimes p \otimes \delta), v) \\
= \quad & \{\, \delta \otimes \delta \otimes p = \delta \otimes p \otimes \delta,\ \otimes \text{ distributes over } \oplus \,\} \\
& (o \circ e^{k})\,(p, q, \delta \otimes p \otimes \delta \otimes p, 1_\oplus, \gamma \oplus (q \otimes \delta) \oplus ((\gamma \oplus (q \otimes \delta)) \otimes \delta \otimes p), v) \\
= \quad & \{\, \text{Def.}\,(e) \,\} \\
& (o \circ e^{k+1})\,(p, q, \delta \otimes p, 1_\oplus, \gamma \oplus (q \otimes \delta), v)
\end{aligned}
$$