# Optimizing Compositions of Scans and Reductions in Parallel Program Derivation

Sergei Gorlatch

*University of Passau, D–94030 Passau, Germany*

We prove two transformation rules over popular primitives of parallel programming: the first rule transforms a sequential composition of scan and reduction into a single reduction, the second rule transforms a composition of two scans into a single scan. For the maximum segment sum problem, we formally derive the two previously known parallel solutions: the derivation of the first solution involves one application of the first rule, the second solution is obtained by a repeated rule application.

**Keywords:** Program derivation; Parallel algorithms; Functional programming; Programming calculi; Bird-Meertens formalism; Scan; Maximum segment sum.

## 1 Introduction

We study two popular programming schemas: *scan* (also known as prefix sums, parallel prefix, etc.) and *reduction* (also known as fold). Originally from the functional world [3], they are becoming increasingly popular as primitives of parallel programming. The reasons are that, first, such higher-order combinators are adequate and useful for a broad class of applications [4], second, they encourage well-structured, coarse-grained parallel programming and, third, their implementation in the MPI standard [14] makes the target programs portable across different parallel architectures with predictable performance.

Our contributions are as follows:

- We formally prove two optimization rules: the first rule transforms a sequential composition of scan and reduction into a single reduction, the second rule transforms a composition of two scans into a single scan.
- We apply the first rule in the formal derivation of a parallel algorithm for the maximum segment sum problem and establish a connection between two earlier solutions of that problem, thereby improving one of them.

The proposed transformations fuse two consecutive global communications into one, thus eliminating extra synchronization and improving performance.

## 2 Notation and Basic Results

We adopt the notation of the *Bird-Meertens formalism* (BMF) [3,11] and restrict ourselves to the total functions over non-empty lists. Such functions are defined by giving their value on a one-element list, say $[a]$, and then inductively, using list concatenation $+\!\!+$ as a constructor.

Function application will be denoted by juxtaposition and has the tightest binding. Binary operators are denoted by $\oplus$, $\otimes$, etc. to stress their genericity and are usually assumed to be associative. We exploit *operator sectioning*, providing only one argument for an operator, thus obtaining a unary function:

$$(a \oplus)\, b \;=\; a \,\oplus\, b \quad \text{or} \quad (\otimes\, a)\, b \;=\; b \,\otimes\, a \tag{1}$$

We use the following, introduced informally, first-order functions on lists:

$$\begin{aligned}
\mathit{inits}\,[x_1, x_2, \ldots , x_n] &= [\,[x_1], [x_1, x_2], \ldots , [x_1, x_2, \ldots , x_n]\,] \\
\mathit{tails}\,[x_1, x_2, \ldots , x_n] &= [\,[x_1, x_2, \ldots , x_n], \ldots , [x_{n-1}, x_n], [x_n]\,] \\
\mathit{last}\,[x_1, x_2, \ldots , x_n] &= x_n\,,\;\; \mathit{head}\,[x_1, x_2, \ldots , x_n] \;=\; x_1
\end{aligned}$$

The essence of BMF constitute second-order functions *map* and *red* (for *reduction*) which we introduce informally as well:

$$\begin{aligned}
\mathit{map}\; f\; [x_1, x_2, \ldots , x_n] &= [\, f\; x_1, f\; x_2, \ldots , f\; x_n\,] \\
\mathit{red}\,(\oplus)\,[x_1, x_2, \ldots , x_n] &= x_1 \,\oplus\, x_2 \,\oplus\, \ldots ,\, \oplus\, x_n
\end{aligned}$$

**Definition 1 (Scan)** *Scan has two dual versions, prefix and suffix, which we denote by pref and suf respectively:*

$$\begin{aligned}
\mathit{pref}\,(\oplus)\,[a] &= [a] \\
\mathit{pref}\,(\oplus)\,(x +\!\!+ y) &= \mathit{pref}\,(\oplus)\,x \;+\!\!+\; \mathit{map}\,(\mathit{last}\,(\mathit{pref}\,(\oplus)\,x)\,\oplus)\,(\mathit{pref}\,(\oplus)\,y)
\end{aligned} \tag{2}$$

$$\begin{aligned}
\mathit{suf}\,(\oplus)\,[a] &= [a] \\
\mathit{suf}\,(\oplus)\,(x \,+\!\!+\, y) &= \mathit{map}\,(\oplus\; \mathit{head}\,(\mathit{suf}\,(\oplus)\,y))\,(\mathit{suf}\,(\oplus)\,x) \;+\!\!+\; \mathit{suf}\,(\oplus)\,y
\end{aligned} \tag{3}$$

We illustrate these functions on an argument list of three elements:

$$\begin{aligned}
\mathit{pref}\,(\oplus)\,[x_1, x_2, x_3] &= [x_1, x_1 \,\oplus\, x_2, x_1 \,\oplus\, x_2 \,\oplus\, x_3] \\
\mathit{suf}\,(\oplus)\,[x_1, x_2, x_3] &= [x_1 \,\oplus\, x_2 \,\oplus\, x_3, x_2 \,\oplus\, x_3, x_3]
\end{aligned}$$

Parallelism is captured in BMF by the notion of homomorphism.

**Definition 2** *A list function $h$ is a homomorphism iff there exists a binary operator $\circledast$ such that, for all lists $x$ and $y$:*

$$h\,(x +\!\!+ y) \;=\; h\,x \;\circledast\; h\,y \tag{4}$$

In words: the value of $h$ on the concatenated list can be computed by the *combine operator* $\circledast$ from the values of $h$ on the pieces of the list. The computations of $h\,x$ and $h\,y$ can be carried out in parallel, and $\circledast$ is associative.

**Theorem 3 (Bird [3])** *Function $h$ on lists is a homomorphism iff:*

$$h \;=\; red\,(\circledast) \;\circ\; map\, f \tag{5}$$

*where, for an element $a$, $f\ a \;=\; h\,[a]$, and $\circledast$ is from* (4).

The theorem provides us with a standard parallel implementation (5) for a homomorphism, as soon as we know $\circledast$ and $f$. From (2)-(3) follows that both scan functions are homomorphisms: e.g., the combine operator of $pref\,(\oplus)$ is defined by $u \circledast v \;\stackrel{\text{def}}{=}\; u \;+\!\!\!+\; map\,((last\, u) \oplus)\; v$.

## 3 Composition of Scan and Reduction

BMF is a program derivation framework, where expressions are built using *backward* functional composition, $\circ$. Expressions are transformable by applying equational rules, some of which are presented here:

$$map\,(f \circ g) \;=\; map\, f \;\circ\; map\, g \tag{6}$$
$$pref\,(\oplus) \;=\; map\,(red\,(\oplus)) \;\circ\; inits \tag{7}$$
$$suf\,(\oplus) \;=\; map\,(red\,(\oplus)) \;\circ\; tails \tag{8}$$
$$inits \;\circ\; map\, f \;=\; map\,(map\, f) \;\circ\; inits \tag{9}$$
$$inits \;\circ\; pref\,(\oplus) \;=\; map\,(pref\,(\oplus)) \;\circ\; inits \tag{10}$$
$$map\, f \;\circ\; red\,(+\!\!\!+) \;=\; red\,(+\!\!\!+) \;\circ\; map\,(map\, f) \tag{11}$$
$$red\,(\oplus) \;\circ\; red\,(+\!\!\!+) \;=\; red\,(\oplus) \;\circ\; map\,(red\,(\oplus)) \tag{12}$$

Program derivation starts with an intuitive, obviously correct but probably inefficient version of an algorithm and proceeds by applying rules like (6)-(12), until an efficient algorithm is obtained.

We are interested in finding a good parallel implementation of a composition, consisting of a scan function, followed by a reduction:

$$prefred\,(\oplus, \otimes) \;\stackrel{\text{def}}{=}\; red\,(\oplus) \;\circ\; pref\,(\otimes) \tag{13}$$

Aiming at a parallel implementation, we try to find a homomorphic representation of *prefred*, following definition (4):

$$\begin{array}{ll}
 & prefred\,(\oplus, \otimes)\,(x \;+\!\!\!+\; y) \\
= & \quad \{\ \text{Eq. (13),(2)}\ \} \\
 & red\,(\oplus)\,\Big(\, pref\,(\otimes)\, x \;+\!\!\!+\; map\,(last\,(pref\,(\otimes)\, x) \otimes)\,(pref\,(\otimes)\, y)\,\Big) \\
= & \quad \{\ \text{Definition of } red \text{ and } last\,(pref\,(\otimes)) \;=\; red\,(\otimes)\ \} \\
 & \Big(\, red\,(\oplus)\,(pref\,(\otimes)\, x)\,\Big) \;\oplus\; \Big(\, red\,(\oplus)\,(map\,((red\,(\otimes)\, x) \otimes)\,(pref\,(\otimes)\, y))\,\Big) \\
= & \quad \{\ \text{Eq. (13)}\ \} \\
 & \Big(\, prefred\,(\oplus, \otimes)\, x\,\Big) \;\oplus\; \Big(\, red\,(\oplus)\,(map\,((red\,(\otimes)\, x) \otimes)\,(pref\,(\otimes)\, y))\,\Big)
\end{array}$$

The obtained intermediate expression does not fit format (4), because functions different from *prefred* are applied to $x$ and $y$. This indicates that *prefred* is not directly a homomorphism. To cure that, we "massage" *prefred* into an almost-homomorphism [8], by tupling it with *red* as an auxiliary function:

$$\mathit{prefred}'\,(\oplus,\otimes) \stackrel{\text{def}}{=} (\mathit{prefred}\,(\oplus,\otimes)\,,\ \mathit{red}\,(\otimes)) \tag{14}$$

Note that *prefred*$'$ yields a pair, where *prefred* is the first component:

$$\mathit{prefred}\,(\oplus,\otimes) \;=\; \pi_1 \,\circ\, \mathit{prefred}'\,(\oplus,\otimes) \tag{15}$$

We can proceed with the derivation, aiming at a homomorphic format for *prefred*$'$; we use the intermediate expression obtained thus far and abbreviate for a while the components of *prefred*$'$: $p = \mathit{prefred}\,(\oplus,\otimes)$ and $r = \mathit{red}\,(\otimes)$:

$$\begin{array}{ll}
 & \mathit{prefred}'\,(\oplus,\otimes)\,(x \mathbin{+\!\!+} y) \\
= & \quad \{\ \text{Eq. (14) and abbreviations}\ \} \\
 & (p\,(x \mathbin{+\!\!+} y)\,,\, r\,(x \mathbin{+\!\!+} y)) \\
= & \quad \{\ \text{Intermediate expression for } p\text{, and definition of } \mathit{red} \text{ for } r\ \} \\
 & \Big(\, p\,x \,\oplus\, \mathit{red}\,(\oplus)\,\Big(\mathit{map}\,(r\,x\,\otimes)\,(\mathit{pref}\,(\otimes)\,y)\Big)\,,\, r\,x \,\otimes\, r\,y\,\Big)
\end{array}$$

If we require that $\otimes$ *distributes forwards over* $\oplus$, i.e. $(a \otimes b) \oplus (a \otimes c) = a \otimes (b \oplus c)$, whose consequence is:

$$\mathit{red}\,(\oplus) \,\circ\, \mathit{map}\,(a\,\otimes) \;=\; (a\,\otimes) \,\circ\, \mathit{red}\,(\oplus) \tag{16}$$

then the first component of the pair can be transformed further:

$$\begin{array}{ll}
p\,x \,\oplus\, (\mathit{red}\,(\oplus) \,\circ\, \mathit{map}\,(r\,x\,\otimes))\,(\mathit{pref}\,(\otimes)\,y) & \{\ \text{Eq. (16)}\ \} \\
= p\,x \,\oplus\, ((r\,x\,\otimes) \,\circ\, \mathit{red}\,(\oplus))\,(\mathit{pref}\,(\otimes)\,y) & \{\ \text{Eq. (13)}\ \} \\
= p\,x \,\oplus\, (r\,x\,\otimes)\,(p\,y) & \{\ \text{Eq. (1)}\ \} \\
= p\,x \,\oplus\, (r\,x \,\otimes\, p\,y) &
\end{array}$$

Therefore, *prefred*$'$ is a homomorphism, and from Theorem 3 follows that $\mathit{prefred}' = \mathit{red}(\circledast) \,\circ\, \mathit{map}\,f$, where

$$f\ a \;=\; \mathit{prefred}'\,[a] \;=\; (\mathit{prefred}\,[a], \mathit{red}\,[a]) \;=\; (a, a) \stackrel{\text{def}}{=} \mathit{pair}\ a \tag{17}$$

$$(p_1, r_1) \circledast (p_2, r_2) \;=\; (p_1 \oplus (r_1 \otimes p_2)\,, r_1 \otimes r_2) \stackrel{\text{def}}{=} (p_1, r_1)\,\langle\oplus,\otimes\rangle\,(p_2, r_2) \tag{18}$$

Here, two new notations have been introduced: *pair* for the function (17) which yields a pair for an element, and $\langle\oplus,\otimes\rangle$ for an operation, built from $\oplus$ and $\otimes$ and defined on pairs according to (18).

A similar derivation for the composition of *suf* with *red* requires in (19) that $\otimes$ *distributes backwards over* $\oplus$, and introduces notation $<\oplus,\otimes>$ in (20):

$$\mathit{red}\,(\oplus) \,\circ\, \mathit{map}\,(\otimes\,a) \;=\; (\otimes\,a) \,\circ\, \mathit{red}\,(\oplus) \tag{19}$$

$$(s_1, r_1) < \oplus, \otimes > (s_2, r_2) \stackrel{\text{def}}{=} ((s_1 \otimes r_2) \oplus s_2\,, r_1 \otimes r_2) \tag{20}$$

Therefore, we have proved that a composition of scan and reduction can be transformed into a single reduction:

**Theorem 4 (Prefix or Suffix, followed by Reduction)** *For arbitrary binary, associative operators* $\oplus$ *and* $\otimes$*:*

– *if* $\otimes$ *distributes over* $\oplus$ *then*

$$red\,(\oplus) \;\circ\; pref\,(\otimes) \;=\; \pi_1 \;\circ\; red\,(\langle\!\langle \oplus, \otimes \rangle\!\rangle) \;\circ\; map\,pair \tag{21}$$

– *if* $\otimes$ *distributes backwards over* $\oplus$ *then*

$$red\,(\oplus) \;\circ\; suf\,(\otimes) \;=\; \pi_1 \;\circ\; red\,(<\!\oplus, \otimes\!>) \;\circ\; map\,pair \tag{22}$$

*where pair,* $\langle \oplus, \otimes \rangle$, $<\oplus, \otimes>$ *are defined by* (17),(18), (20) *respectively.*

## 4 Composition of Scans

In this section, we optimize a composition of two scans:

$$\begin{array}{lr}
\;\; pref\,(\oplus) \;\circ\; pref\,(\otimes) & \{\text{ Eq. (7) }\} \\
= map\,(red\,(\oplus)) \;\circ\; inits \;\circ\; pref\,(\otimes) & \{\text{ Eq. (10) }\} \\
= map\,(red\,(\oplus)) \;\circ\; map\,(pref\,(\otimes)) \;\circ\; inits & \{\text{ Eq. (6) }\} \\
= map\,(red\,(\oplus) \;\circ\; pref\,(\otimes)) \;\circ\; inits & \{\text{ Th. 4 }\} \\
= map\,(\pi_1 \;\circ\; red\,(\langle\!\langle \oplus, \otimes \rangle\!\rangle) \;\circ\; map\,pair) \;\circ\; inits & \{\text{ Eq. (6) }\} \\
= map\,\pi_1 \;\circ\; map\,(red\,(\langle\!\langle \oplus, \otimes \rangle\!\rangle)) \;\circ\; map\,(map\,pair) \;\circ\; inits & \{\text{ Eq. (9) }\} \\
= map\,\pi_1 \;\circ\; map\,(red\,(\langle\!\langle \oplus, \otimes \rangle\!\rangle)) \;\circ\; inits \;\circ\; map\,pair & \{\text{ Eq. (7) }\} \\
= map\,\pi_1 \;\circ\; pref\,(\langle\!\langle \oplus, \otimes \rangle\!\rangle) \;\circ\; map\,pair &
\end{array}$$

A similar calculation can be done for a composition of two functions *suf*. We make use of Theorem 4, whose assumptions about the distributivity of the involved operators have to be passed over to the new theorem:

**Theorem 5 (Composition of Scans )** *For arbitrary binary, associative operators* $\oplus$ *and* $\otimes$*:*

– *if* $\otimes$ *distributes over* $\oplus$ *then*

$$pref\,(\oplus) \;\circ\; pref\,(\otimes) \;=\; map\,\pi_1 \;\circ\; pref\,(\langle\!\langle \oplus, \otimes \rangle\!\rangle) \;\circ\; map\,pair \tag{23}$$

– *if* $\otimes$ *distributes backwards over* $\oplus$ *then*

$$suf\,(\oplus) \;\circ\; suf\,(\otimes) \;=\; map\,\pi_1 \;\circ\; suf\,(<\!\oplus, \otimes\!>) \;\circ\; map\,pair \tag{24}$$

*where pair,* $\langle \oplus, \otimes \rangle$, $<\oplus, \otimes>$ *are defined by* (17),(18), (20) *respectively.*

## 5 Application to the Maximum Segment Sum Problem

We consider the famous *maximum segment sum* (*mss*) problem – a *programming pearl* [1], studied by many authors [3,6,11–13]. Given a list of numbers, function *mss* finds a contiguous list segment whose members have the largest sum among all such segments and returns this sum. For example:

$$mss\,[\,2, -4, 2, -1, 6, -3\,] \;=\; 7$$

where the result is contributed by the segment $[2, -1, 6]$.

We start with the following intuitive algorithm:

$$mss \;=\; red\,(\uparrow) \;\circ\; map\,(red\,(+)) \;\circ\; segs \tag{25}$$

which consists of three stages, from right to left:

*segs* : yields the list of all segments of the original list;
$map\,(red\,(+))$ : for each segment, computes the sum of its elements by the reduction with the usual addition, $+$; the list of sums is thus obtained;
$red\,(\uparrow)$ : the largest sum is computed by the reduction with operator $\uparrow$ that computes the maximum of its two arguments.

A constructive method for producing all segments of a list is given by:

$$segs \;=\; red\,(+\!\!+) \;\circ\; map\;tails \;\circ\; inits \tag{26}$$

Specification (25)-(26) is obviously correct but has a poor time complexity – $O\,(n^3)$ for an input list of length $n$, in both sequential and parallel cases [11].

Note that our concrete operators, $\uparrow$ and $+$, are both associative; moreover, $+$ distributes over $\uparrow$ backwards: $(a \uparrow b) + c \;=\; (a + c) \uparrow (b + c)$. Therefore, we can use Theorem 4 for suffix in the following derivation:

$$
\begin{array}{ll}
 & mss \;=\; red\,(\uparrow) \;\circ\; map\,(red\,(+)) \;\circ\; red\,(+\!\!+) \;\circ\; map\;tails \;\circ\; inits \\
= & \quad \{\ \text{Eq. (11)}\ \} \\
 & red\,(\uparrow) \;\circ\; red\,(+\!\!+) \;\circ\; map\,(map\,(red\,(+))) \;\circ\; map\;tails \;\circ\; inits \\
= & \quad \{\ \text{Eq. (12)}\ \} \\
 & red\,(\uparrow) \;\circ\; map\,(red\,(\uparrow)) \;\circ\; map\,(map\,(red\,(+))) \;\circ\; map\;tails \;\circ\; inits \\
= & \quad \{\ \text{Eq. (6)}\ \} \\
 & red\,(\uparrow) \;\circ\; map\;\Big(\,red\,(\uparrow) \;\circ\; map\,(red\,(+)) \;\circ\; tails\,\Big) \;\circ\; inits \\
= & \quad \{\ \text{Eq. (8)}\ \} \\
 & red\,(\uparrow) \;\circ\; map\;\Big(\,red\,(\uparrow) \;\circ\; suf\,(+)\,\Big) \;\circ\; inits \\
= & \quad \{\ \text{Theorem 4 for suffix, with } \otimes = + \text{ and } \oplus = \uparrow\ \} \\
 & red\,(\uparrow) \;\circ\; map\;\Big(\,\pi_1 \;\circ\; red\,\Big(<\uparrow, +>\Big) \;\circ\; map\;pair\,\Big) \;\circ\; inits \\
= & \quad \{\ \text{Eq. (6),(9)}\ \}
\end{array}
$$

$$red\,(\uparrow)\ \circ\ map\,\pi_1\ \circ\ map\,(red\,\big(<\uparrow,+>\big))\ \circ\ inits\ \circ\ map\,pair$$
$$=\quad \{\ \text{Eq. (7)}\ \}$$
$$red\,(\uparrow)\ \circ\ map\,\pi_1\ \circ\ pref\,\big(<\uparrow,+>\big)\ \circ\ map\,pair$$

We have arrived at an expression for $mss$ which can be implemented in time $O\,(n)$ sequentially and $O\,(\log n)$ in parallel. Derivation [5] stops here, but we can actually advance further:

– Extend operator $\uparrow$ to pairs: $(a,b)\Uparrow(c,d)\ \stackrel{\text{def}}{=}\ (a\uparrow c\,,b\uparrow d)$ and note that $\Uparrow$ is associative and also that:

$$\pi_1\ \circ\ red\,(\Uparrow)\ =\ red\,(\uparrow)\ \circ\ map\,\pi_1 \tag{27}$$

– Note that $<\uparrow,+>$ distributes forwards over $\Uparrow$, which enables an application of Theorem 4 for prefix, with $\otimes=<\uparrow,+>$ and $\oplus=\Uparrow$.

The derivation continues from the previous point as follows:

$$\begin{array}{ll}
\quad mss & \{\ \text{Eq. (27)}\ \}\\
=\pi_1\ \circ\ red\,(\Uparrow)\ \circ\ pref\,\big(<\uparrow,+>\big)\ \circ\ map\,pair & \{\ \text{Theorem 4}\ \}\\
=\pi_1\ \circ\ \pi_1\ \circ\ red\,\big(\langle\Uparrow,<\uparrow,+>\rangle\big)\ \circ\ map\,pair\ \circ\ map\,pair & \{\ \text{Eq. (6)}\ \}\\
=\pi_1\ \circ\ \pi_1\ \circ\ red\,\big(\langle\Uparrow,<\uparrow,+>\rangle\big)\ \circ\ map\,(pair\ \circ\ pair) &
\end{array}$$

Denoting $\pi_1^2\ \stackrel{\text{def}}{=}\ \pi_1\ \circ\ \pi_1$ and $pair^2\ \stackrel{\text{def}}{=}\ pair\ \circ\ pair$, we obtain the following target program for $mss$:

$$mss\ =\ \pi_1^2\ \circ\ red\,\big(\langle\Uparrow,<\uparrow,+>\rangle\big)\ \circ\ map\,(pair^2) \tag{28}$$

Program (28) consists of three stages, where the first stage creates a pair of pairs (quadruple) of each element and the third stage picks the first component of a quadruple. Each of these stages obviously requires constant parallel time.

The central stage of (28) is the reduction with operator $\langle\Uparrow,<\uparrow,+>\rangle$, whose behaviour on pairs of pairs can be expressed in terms of $\uparrow$ and $+$, if we recall the definitions of the $\langle\,\rangle$, $<>$ and $\Uparrow$:

$$\begin{array}{l}
\Big(\ (r_1,s_1),(t_1,u_1)\ \Big)\ \langle\Uparrow,<\uparrow,+>\rangle\ \Big(\ (r_2,s_2),(t_2,u_2)\ \Big)=\\
\qquad\Big(\ \Big(r_1\uparrow r_2\uparrow(t_1+s_2)\,,s_1\uparrow(u_1+s_2)\Big)\,,\Big(t_2\uparrow(t_1+u_2)\,,u_1+u_2\Big)\ \Big)
\end{array} \tag{29}$$

Interestingly enough, we have arrived exactly at the quadruple solution with parallel time complexity $O\,(\log n)$ which was proposed in [6] and later obtained in a systematic way by generalizing two sequential programs in [8]. The components of a quadruple have a problem-specific meaning – maximum initial segment sum, total sum, etc. – but this meaning has not been called for in our derivation.

## 6 Discussion

Our optimization rules and derivations demonstrate the elegance and power of the calculational BMF approach in parallel programming. A common ground for homomorphism fusion is given by the category-theoretical results on catamorphisms [10]. Our Theorems 4 and 5 provide a direct method for testing the applicability of the transformations and computing their results.

A version of Theorem 4 in the sequential setting was proved and used in [2,13]. In [5], a version of *red* $(<\oplus, \otimes>)$, called *recur-reduce*, is used for parallelizing linear recurrences. An approach dual to ours is taken in our own group [15]: a *foldl* with an analogue of $\langle\oplus, \otimes\rangle$ is represented as a composition of four functions: functional products of scan and reduction with the identity function, and two regrouping operations. Our transformation aims directly at the scan-reduction composition and is arguably more convenient for use in the practical programming with the reduction and scan primitives. A systematic process of homomorphism construction yields the transformations together with their proofs, rather than first inventing a transformation and then proving its correctness ([5,15] show no formal proofs).

The transformation of two scans in Theorem 5 is, to the best of our knowledge, new; the target expression is similar to the *recur-prefix* introduced in [5].

The pragmatics of our two optimization rules is to transform a composition of two global communication operations into one global communication, with a more complex basic operator. Although not improving the asymptotic time complexity, such fusion eliminates a global synchronization barrier between the two stages. That can provide a substantial performance improvement on contemporary parallel machines, especially taking into account recent advances in the hardware implementation of the MPI-like global operations [9].

We have given yet another derivation for the maximum segment sum problem. In the sequential setting, related work was done in [2,13]. We have established a connection between two parallel solutions: an application of Theorem 4 for suffix leads to a solution similar to [5], which, by a further application of Theorem 4 for prefix, is improved and transformed into the solution from [6,12]. Our derivation combines the advantages of both solutions: it is carried out without thinking about the meaning of the involved operators, and results in just one global parallel operation.

The derived target programs exhibit the maximal parallelism of the specifications and require a linear number of processors; together with the logarithmic parallel time, this yields the sub-optimal parallel cost [9]. It is more realistic to consider an arbitrary bounded number of processors, which leads to the cost-optimal solutions for homomorphisms [7].

### Acknowledgement

Discussions with Christoph Wedler helped me to complete this work.

## References


[1] J. Bentley. Programming pearls. *Communications of the ACM*, 27:865–871, 1984.

[2] R. Bird. Algebraic identities for program calculation. *The Computing Journal*, 32:122–126, 1989.

[3] R. S. Bird. Lectures on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computing Science*, NATO ASI Series F: Computer and Systems Sciences. Vol. 55, pages 151–216. Springer Verlag, 1988.

[4] G. Blelloch. Scans as primitive parallel operations. *IEEE Trans. on Computers*, 38(11):1526–1538, November 1989.

[5] W. Cai and D. Skillicorn. Calculating recurrences using the Bird-Meertens formalism. *Parallel Processing Letters*, 5(2):179–190.

[6] M. Cole. Parallel programming with list homomorphisms. *Parallel Processing Letters*, 5(2):191–204, 1994.

[7] S. Gorlatch. Systematic efficient parallelization of scan and other list homomorphisms. In L. Bougé et al., editors, *Parallel Processing. Euro-Par'96*, Lecture Notes in Computer Science 1124, pages 401–408. Springer-Verlag, 1996.

[8] S. Gorlatch. Systematic extraction and implementation of divide-and-conquer parallelism. In H. Kuchen and D. Swierstra, editors, *Programming Languages: Implementation, Logics and Programs. PLILP'96*, Lecture Notes in Computer Science 1140, pages 274–288. Springer-Verlag, 1996.

[9] V. Kumar et al. *Introduction to Parallel Computing.* Benjamin/Cummings Publ., 1994.

[10] E. Meijer, M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In J. Hughes, editor, *FPCA'91 Proceedings*, pages 124–144. Springer-Verlag, 1991.

[11] D. Skillicorn. *Foundations of Parallel Programming.* Cambridge U. P., 1994.

[12] D. Smith. Applications of a strategy for designing divide-and-conquer algorithms. *Science of Computer Programming*, 8(3):213–229, 1987.

[13] D. Swierstra and O. de Moor. Virtual data structures. In B. Moeller, H. Partsch, and S. Schuman, editors, *Formal Program Development*, Lecture Notes in Computer Science 755, pages 355–371.

[14] D. Walker. The design of a standard message passing interface for distributed memory concurrent computers. *Parallel Computing*, 20:657–673, 1994.

[15] C. Wedler and C. Lengauer. Parallel implementations of combinations of broadcast, reduction and scan. In G. Agha and S. Russo, editors, *Proc. 2nd Int. Workshop on Software Engineering for Parallel and Distributed Systems (PDSE'97)*, 1997.