Parallel Execution of the Saturated Reductions

Benoît Dupont de Dinechin, Christophe Monat, Fabrice Rastello

To cite this version:

Benoît Dupont de Dinechin, Christophe Monat, Fabrice Rastello. Parallel Execution of the Saturated Reductions. [Research Report] LIP RR-2001-28, Laboratoire de l'informatique du parallélisme. 2001, 2+15p. hal-02101824

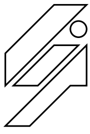
HAL Id: hal-02101824

https://hal-lara.archives-ouvertes.fr/hal-02101824

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

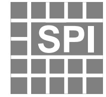
L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 8512

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

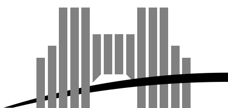


Parallel Execution of the Saturated Reductions

Benoît Dupont de Dinechin
Christophe Monat
Fabrice Rastello

July 2001

Research Report N° 2001-28



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

INRIA

Parallel Execution of the Saturated Reductions

Benoît Dupont de Dinechin
Christophe Monat
Fabrice Rastello

July 2001

Abstract

This report addresses the problem of improving the execution performance of saturated reduction loops on fixed-point instruction-level parallel Digital Signal Processors (DSPs). We first introduce "bit-exact" transformations, that are suitable for use in the ETSI and the ITU speech coding applications. We then present "approximate" transformations, the relative precision of which we are able to compare. Our main results rely on the properties of the saturated arithmetic.

Keywords: Saturated Reductions, Fixed-Point Arithmetic, Fractional Arithmetic, ETSI / ITU Speech Coding, Parallel Reductions, Instruction Level Parallelism

Résumé

Ce rapport traite de la parallélisation des réductions saturées sur les processeurs DSP à arithmétique virgule fixe. Nous proposons dans un premier temps des transformations "bit-exactes", appropriée aux applications de codage de parole ETSI et ITU. Puis nous présentons plusieurs transformations "approchées", que nous comparons. Nos principales contributions sont basées sur les propriétés de l'arithmétique saturée.

Mots-clés: Réductions saturées, Arithmétique à virgule-fixe, Arithmétique fractionnaire, Codeurs de parole ETSI / ITU, Réductions parallèles, Parallélisme au niveau des instructions.

Parallel Execution of the Saturated Reductions

Benoît Dupont de Dinechin *
Christophe Monat †
Fabrice Rastello ‡

July 2001

1 Introduction

The latest generation of fixed-point Digital Signal Processors (DSPs), that comprises the Texas Instruments C6200 and C6400 series [8], the StarCore SC140 [6], and the STMicroelectronics ST120 [7], rely on instruction-level parallelism exploitation, and fractional arithmetic support, in order to deliver high-performance at low cost on telecommunication and mobile applications.

In particular, the instruction set of these so-called DSP–MCUs [3] is tailored to the efficient implementation of standardized speech coding algorithms such as those published by the ITU (International Telecommunication Union) [5], and the ETSI (European Telecommunication Standards Institute) [4]. The widely used ETSI / ITU speech coding algorithms include:

ETSI EFR-5.1.0 The Enhanced Full Rate algorithm, used by the European GSM mobile telephone systems.

ETSI AMR-NB The Adaptative Multi Rate Narrow Band, that will be used in the 3GPP/UMTS mobile telephone systems.

ITU G.723.1, ITU G.729 Speech coders used in Voice over IP (VoIP), Voice over Network (VoN), and for H.324 and H.323 applications.

In the ETSI and the ITU reference implementations of these algorithms, data are stored as 16-bit integers under the $Q_{1.15}$ fractional representation, and are operated as 32-bit integers under the $Q_{1.31}$ fractional representation. The $Q_{n.m}$ fractional representation interprets a $n+m$ bit integer x as [3]:

$$Q_{n.m}(x) = -2^{n-1}x_{n-1+m} + \sum_{k=0}^{n-2+m} 2^{k-m}x_k$$

In other words, $Q_{n.m}$ is the two's complement integer representation on $n+m$ bits, scaled by 2^{-m}, so the range of a $Q_{n.m}$ number is $[-2^{n-1}, 2^{n-1} - 2^{-m}]$.

Two's complement addition of $Q_{n.m}$ numbers yields a $Q_{n+1.m}$ number. Two's complement multiplication of $Q_{n.m}$ numbers yields a $Q_{2n.2m}$ number with the two upper bits identical except in the $-2^{n-1} \times -2^{n-1} = 2^{2n-2}$ case. This multiplication result fits into a $Q_{2n-1.2m+1}$ number, after saturation of 2^{2n-2} to $2^{2n-2} - 2^{-2m-1}$. In particular, the $Q_{1.m}$ representations ($n = 1$) are widely used because their multiplication exactly fits $Q_{1.2m+1}$ numbers, except in the -1×-1 case where the relative saturation error is 2^{-2m-1}.

Because the ETSI and the ITU reference implementations operate on the $Q_{1.15}$ and the $Q_{1.31}$ fractional representations, a saturation is involved every time two $Q_{1.31}$ numbers are added into a $Q_{1.31}$ number, and every time two $Q_{1.15}$ numbers are multiplied into a $Q_{1.31}$ number.

*STMicroelectronics. 12 rue Jules Horowitz, BP217, F-38019 Grenoble Cedex France. Benoit.Dupont-de-Dinechin@st.com
†STMicroelectronics. 12 rue Jules Horowitz, BP217, F-38019 Grenoble Cedex France. Christophe.Monat@st.com
‡STMicroelectronics. 12 rue Jules Horowitz, BP217, F-38019 Grenoble Cedex France. Fabrice.Rastello@ens-lyon.fr

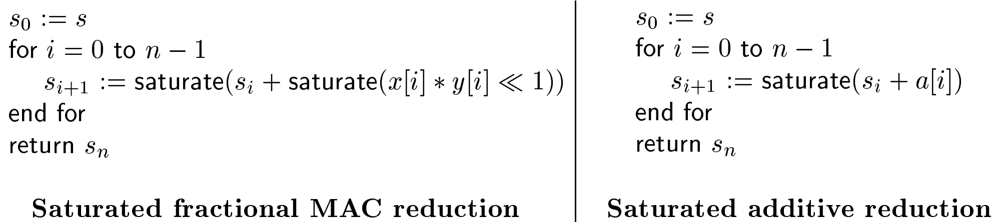


Figure 1: Saturated reductions: motivating examples.

Let the saturate operator be defined on a 32-bit number s interpreted as a two's complement integer:

$$\begin{aligned} \mathsf{saturate}(s) &\stackrel{\text{def}}{=} \begin{cases} \textbf{if } s > \mathsf{max32} \textbf{ then } \mathsf{max32} \\ \textbf{else if } s < \mathsf{min32} \textbf{ then } \mathsf{min32} \\ \textbf{else } s \end{cases} \\ \mathsf{max32} &\stackrel{\text{def}}{=} 2^{31} - 1 \\ \mathsf{min32} &\stackrel{\text{def}}{=} -2^{31} \end{aligned}$$

In the ETSI and the ITU vocoders, a significant percentage of the computations is spent on *saturated reductions* loops like those displayed in figure 1. In this figure, s and $a[i]$ are 32-bit $Q_{1.31}$ numbers, while $x[i]$ and $y[i]$ are 16-bit $Q_{1.15}$ numbers. Subscripted symbols like s_i are temporary variables indexed by the loop iteration number i, in order to reference their value in proofs. In real programs, such temporaries would be mapped to a single variable.

In the saturated fractional MAC reduction code of figure 1, the $x[i] * y[i]$ product is a 32-bit $Q_{2.30}$ number, which is first shifted left one bit in order to align the fractional point. The result is a 33-bit $Q_{2.31}$ number, which is saturated back to a 32-bit $Q_{1.31}$ number. This value resulting from $\mathsf{saturate}(x[i] * y[i] \ll 1)$ is then added to s, to yield a 33-bit $Q_{2.31}$ number, which is saturated again to 32-bit, yielding the new value of s under the $Q_{1.31}$ representation.

In order to efficiently execute speech coding applications under the requirement of "bit-exactness" with the the ETSI and the ITU reference implementations, a DSP must support fractional arithmetic in its instruction set. As a matter of fact, today's leading fixed-point DSPs offer *multiply-accumulate* (MAC) instructions, including fractional multiply-accumulates that compute $s := \mathsf{saturate}(s + \mathsf{saturate}(x[i] * y[i] \ll 1))$ and its variants in a single cycle.

On the new DSP–MCUs, exposing more instruction-level parallelism than a single MAC per cycle is required in order to reach the peak performances. Indeed, these processors are able to execute two (TI C6200, STM ST120) to four (TI C6400, StarCore SC140) MACs per cycle. Unfortunately, saturated additive reductions are neither commutative nor associative, unlike modular integer additive reductions. Therefore, a main issue when optimizing the ETSI and the ITU reference implementations onto a particular DSP–MCU is to expose instruction-level parallelism on saturated reduction loops.

In this report, we discuss several techniques that enable the parallel execution of saturated reduction loops on the new DSP–MCUs. In section 2, we present the "bit exact" transformations that are suitable for use in the ETSI / ITU speech coding algorithms, but require a 4-MAC DSP. In section 3, we study the relative correctness of approximate transformations of saturated reduction loops that are commonly used on 2-MAC DSPs.

2 Bit-Exact Parallel Saturated Reductions

2.1 Compiler Optimization of the ETSI and the ITU Codes

In the ETSI and the ITU reference implementations of speech coding algorithms, all the $Q_{1.15}$ and $Q_{1.31}$ arithmetic operations are available as the functions known as the *basic operators* (files `basicop2.c` and `basic_op.h` in the ETSI codes). Among the basic operators, the most heavily used are (ordered by decreasing dynamic execution counts measured on the ETSI EFR-5.1.0):

```
for (i = 0; i < lg; i++) {
  s = L_mult(x[i], a[0]);
  for (j = 1; j <= M; j++) {
    s = L_mac(s, a[j], x[i - j]);
  }
  s = L_shl(s, 3);
  y[i] = round(s);
}
```

(residu)

```
for (n = 0; n < L; n++) {
  s = 0;
  for (i = 0; i <= n; i++) {
    s = L_mac(s, x[i], h[n - i]);
  }
  s = L_shl(s, 3);
  y[n] = extract_h(s);
}
```

(convolve)

Figure 2: Saturated reductions: original code.

$$
\begin{array}{lcl}
\mathsf{add}(x,y) & \stackrel{\text{def}}{=} & \mathsf{saturate}(x+y) \gg 16 \\
\mathsf{L_mac}(s,x,y) & \stackrel{\text{def}}{=} & \mathsf{saturate}(s+\mathsf{saturate}(x*y \ll 1)) \\
\mathsf{mult}(x,y) & \stackrel{\text{def}}{=} & \mathsf{saturate}(x*y \ll 1) \gg 16 \\
\mathsf{L_msu}(s,x,y) & \stackrel{\text{def}}{=} & \mathsf{saturate}(s-\mathsf{saturate}(x*y \ll 1)) \\
\mathsf{L_mult}(x,y) & \stackrel{\text{def}}{=} & \mathsf{saturate}(x*y \ll 1) \\
\mathsf{round}(s) & \stackrel{\text{def}}{=} & \mathsf{saturate}(s+2^{15}) \gg 16
\end{array}
$$

In these expressions, x and y are 16-bit numbers under the $Q_{1.15}$ representation, while s and t are 32-bit numbers under the $Q_{1.31}$ representation.

When porting an ETSI / ITU reference implementation to a particular DSP, the first step is to redefine the basic operators as *intrinsic functions*, that is, functions known to the target C compiler, and inlined into one or a few instructions of the target processor. Efficient inlining is compiler challenging, as virtually all the ETSI / ITU basic operators have a side-effect on a C global variable named `Overflow`, which is set whenever saturate effectively saturates its argument. Compiler data-flow analysis is used to isolate the reductions whose side-effects can be safely ignored.

Precisely, on the ST120 DSP–MCU processor by STMicroelectronics [7], the side-effects of hardware saturation and integer overflow are accumulated into sticky status registers[1]. The main optimization that the C compiler performs after mapping the ESTI / ITU basic operators to the ST120 instructions, is to dead-code eliminate the useless transfers of values between the `Overflow` variable, and the ST120 sticky status registers.

Once efficient inlining of the ESTI / ITU basic operators is achieved, the performance bottlenecks are identified in order to trigger more aggressive compiler optimizations. On the ESTI / ITU speech coding algorithms, many of these bottlenecks involve saturated reduction loops. Typical examples of such loops, from the EFR-5.1.0 vocoder, are illustrated in figure 2. In these cases, parallel execution can be achieved without introducing any overhead, thanks to the *unroll-and-jam* compiler optimization.

Unroll-and-jam [1] can be described as outer loop unrolling, followed by the loop fusion of the resulting inner loops. The main issues with this transformation are checking the validity of the inner loop fusion, and dealing with iteration bounds of the inner loop that are outer loop variant. Unroll-and-jam of the codes of figure 2 is illustrated in figure 3. In the **residu** code, the compiler Inter-Procedural Analysis (IPA) infers that `lg` is even, and that `y` does not alias `a` or `x`. Likewise in the **convolve** code, where the IPA infers that `L` is even, and that `y` is alias-free. Thanks to these informations, remainder code for the outer loop unrolling is avoided, and the inner loop fusion is found legal. Unlike **residu**, the **convolve** loop nest has a triangular iteration domain, so its inner loop fusion generates residual computations.

Unroll-and-jam of saturated reduction loops is not always an option. Either the memory dependencies carried by the outer loop prevent the fusion of the inner loops after unrolling, such as in the **syn_filt** code found in the ETSI EFR-5.1.0 and the ETSI AMR-NB. Or there are no outer loops suitable for unrolling. In such cases, parallel execution can still be achieved, by using the arithmetic properties of the saturated additive reduction.

[1] Registers that are only set as side-effect of instructions, and that must be reset with a dedicated instruction. Because of the stickyness property, updates to these status registers can proceed in parallel.

```
for (i = 0; i < lg/2; i++) {
  short i_e = 2 * i ;
  short i_o = 2 * i + 1 ;
  int s_e = L_mult(x[i_e], a[0]);
  int s_o = L_mult(x[i_o], a[0]);
  for(j = 1 ; j <= M; j++) {
    s_e = L_mac(s_e, a[j], x[i_e - j]);
    s_o = L_mac(s_o, a[j], x[i_o - j]);
  }
  s_e = L_shl(s_e, 3);
  s_o = L_shl(s_o, 3);
  y[i_e] = round(s_e) ;
  y[i_o] = round(s_o) ;
}
```

(residu)

```
for (n = 0; n < L/2; n++) {
  short n_e = 2 * n;
  short n_o = 2 * n + 1 ;
  int s_e = 0;
  int s_o = 0;
  for (i = 0; i <= n_e; i++) {
    s_e = L_mac(s_e, x[i], h[n_e - i]);
    s_o = L_mac(s_o, x[i], h[n_o - i]);
  }
  s_o = L_mac(s_o, x[n_o], h[0]);
  s_e = L_shl(s_e, 3);
  s_o = L_shl(s_o, 3);
  y[n_e] = extract_h(s_e);
  y[n_o] = extract_h(s_o);
}
```

(convolve)

Figure 3: Saturated reductions: after unroll-and-jam.

$u_0 := s$
$v_0 := 0$
$min_0 :=$ min32
$max_0 :=$ max32
for $i = 0$ to $\frac{n}{2} - 1$
  $u_{i+1} :=$ saturate$(u_i + a[i])$
  $v_{i+1} := v_i + a[i + \frac{n}{2}]$
  $min_{i+1} :=$ saturate$(min_i + a[i + \frac{n}{2}])$
  $max_{i+1} :=$ saturate$(max_i + a[i + \frac{n}{2}])$
end for
return $\mathsf{clip}_{min_{\frac{n}{2}}}^{max_{\frac{n}{2}}}(u_{\frac{n}{2}} + v_{\frac{n}{2}})$

Program 1

```
int u = s;
long v = 0;
int min = INT_MIN, max = INT_MAX;
for (i = 0; i < n/2; i++) {
  u = L_mac(u, x[i], y[i]);
  v += L_mult(x[i+n/2], y[i+n/2]);
  min = L_mac(min, x[i+n/2], y[i+n/2]);
  max = L_mac(max, x[i+n/2], y[i+n/2]);
}
v += u;
if (v > max) return max;
if (v < min) return min;
return v;
```

Program 2

Figure 4: Saturated reductions: parallelized code with one 40-bit accumulation.

2.2 Exploitation of a 4-MAC DSP with 40-bit Accumulators

We now introduce a first technique, based on the arithmetic properties of the saturated additive reduction, that enables the "bit-exact" parallel execution of the saturated reductions. This techniques requires a DSP that executes four MAC per cycle, to achieve an effective throughput of two iterations of the original saturated reduction loop per cycle. The pseudo-code and the C code that implement this technique are displayed in figure 4. Program 2 assumes the data-type mapping of the TI C6000 and the STM ST120 C compilers: `long` integers are 40-bit, `int`egers are 32-bit, and `short` integers are 16-bit.

Let us first consider the two programs in figure 5, that compute a saturated reduction over n values $a[i]$. The clip operator is such that saturate $\equiv \mathsf{clip}_{\mathsf{min32}}^{\mathsf{max32}}$:

$$\mathsf{clip}_l^h(s) \stackrel{\text{def}}{=} \begin{cases} \quad\;\;\textbf{if } l > h \textbf{ then } \perp \\ \textbf{else if } s > h \textbf{ then } h \\ \textbf{else if } s < l \textbf{ then } l \\ \textbf{else } s \end{cases}$$

Theorem 1 *In figure 5, if $min_0 \leq s \leq max_0$, then Program 3 and Program 4 compute the same result.*

```
s_0 := s
for i = 0 to n − 1
    s_{i+1} := saturate(s_i + a[i])
end for
return s_n
```

Program 3

```
S_0 := s
min_0 := ...
max_0 := ...
for i = 0 to n − 1
    S_{i+1} := S_i + a[i]
    min_{i+1} := saturate(min_i + a[i])
    max_{i+1} := saturate(max_i + a[i])
end for
return clip_{min_n}^{max_n}(S_n)
```

Program 4

Figure 5: Saturated reductions: equivalent codes when $min_0 \leq s \leq max_0$.

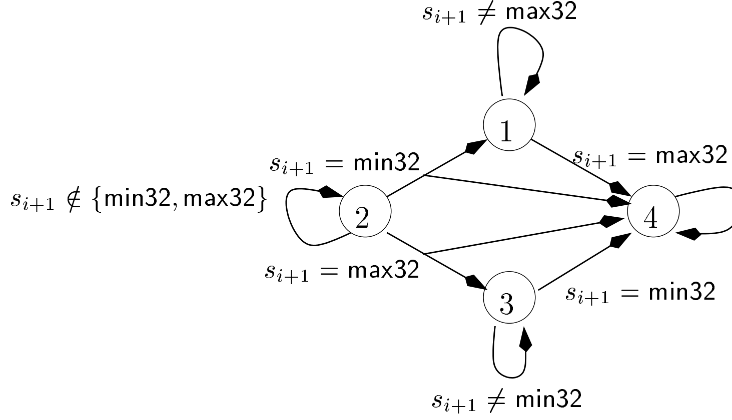


Figure 6: Possible case transitions between the induction steps i and $i+1$ of Theorem 1.

The proof is done by induction on the iteration index i, under the induction hypothesis that s_i in Program 3 equals $\mathsf{clip}_{min_i}^{max_i}(S_i)$ in Program 4. This induction hypothesis is actually equivalent to the four following cases:

$$S_i \leq min_i = s_i < max_i \quad (1)$$
$$\text{or} \quad min_i \leq s_i = S_i \leq max_i \text{ and } min_i \neq max_i \quad (2)$$
$$\text{or} \quad min_i < max_i = s_i \leq S_i \quad (3)$$
$$\text{or} \quad min_i = s_i = max_i \quad (4)$$

Because $min_0 \leq S_0 = s \leq max_0$, $\mathsf{clip}_{min_0}^{max_0}(S_0) = S_0 = s$, while $s_0 = s$, so the induction hypothesis is verified for $i = 0$. The proof is completed by applying Lemma 1 to each of the four cases above, as summarized in Figure 6. □

Lemma 1 *Let x and y be two numbers.*
If $x \leq y$ then $\mathsf{min32} \leq \mathsf{saturate}(x + a[i]) \leq \mathsf{saturate}(y + a[i]) \leq \mathsf{max32}$
If $x \leq y$ and $a[i] \leq 0$ then $x + a[i] \leq \mathsf{saturate}(y + a[i])$.
If $x \leq y$ and $a[i] \geq 0$ then $\mathsf{saturate}(x + a[i]) \leq y + a[i]$.

The proof follows from the definition of saturate. □

Corollary 1 *Program 1 and program 3 compute the same result.*

Let us introduce the following notations:

$$\bigoplus_{i=0}^{m-1}(s, a[i]) \stackrel{\text{def}}{=} \begin{cases} s_0 := s \\ \textsf{for } i = 0 \textsf{ to } m-1 \\ \quad s_{i+1} := \textsf{saturate}(s_i + a[i]) \\ \textsf{end for} \\ \textsf{return } s_m \end{cases} , \text{ and } \bigoplus_{i=0}^{m-1}(a[i]) \stackrel{\text{def}}{=} \bigoplus_{i=0}^{m-1}(0, a[i])$$

At the Program 1 **end for** control point, we have: $u_{\frac{n}{2}} = \bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[i])$, so the result s_n computed by Program 3 equals:

$$s_n = \bigoplus_{i=0}^{n-1}(s, a[i]) = \bigoplus_{i=\frac{n}{2}}^{n-1}\left(\bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[i])\ ,\ a[i]\right) = \bigoplus_{i=\frac{n}{2}}^{n-1}(u_{\frac{n}{2}}, a[i])$$

However, $\textsf{min32} \leq u_{\frac{n}{2}} \leq \textsf{max32}$, so by applying Theorem 1, with S_0 replaced by $u_{\frac{n}{2}}$ and S_i replaced by $u_{\frac{n}{2}} + v_i$ in Program 4, we get the stated result. □

Note The proof of Corollary 1 assumes infinite precision arithmetic for computing S_i in Program 4, as opposed to the 32-bit arithmetic that is used for min_i and max_i. As far as n does not exceed 2^8, `long` integer computations are equivalent to using unlimited precision integer arithmetic. In particular this condition is verified in the ETSI / ITU reference implementations, where the main vector lengths are 40 and 160. On applications where the vector lengths may exceed 2^8, strip-mining of the reduction loop can be used to create inner loops that iterate no more than 2^8.

However, by applying Lemma 2 of the next section, we find that using 33-bit arithmetic in place of infinite precision arithmetic for computing S_i in Program 4 is enough: whenever S_i overflows in 33-bit arithmetic, $min_n = max_n$, so S_n no longer contributes to the end result of Program 4. Thus strip-mining of the reduction loop is not necessary, and Program 2 actually works whatever the loop iteration count n. In addition, Program 2 only requires that `long` integers are larger than 32-bit.

2.3 Exploitation of a 4-MAC DSP with 32-bit Accumulators

One problem with the method of section 2.2 is that it requires a target DSP that computes three saturated 32-bit MACs, plus one non-saturated 40-bit MAC, per cycle. In this section, we show that Program 5 in figure 7 is bit-exact. The corresponding C code in Program 6 achieves an effective throughput of two iterations of the original saturated reduction loop per cycle on a 4-MAC DSP, and only requires 32-bit accumulations: three with saturation, and one that uses 32-bit modular integer arithmetic denoted $\stackrel{32}{+}$.

Theorem 2 *In figure 8, Program 4' and Program 7 compute the same result.*

A first remark is that if $min_n = max_n$, then Program 4' and Program 7 return the same result: $min_n = max_n$. Hence we shall assume $min_n < max_n$, and the proof reduces to Lemma 2, followed by the simple observations:

$$\begin{aligned} (6) \quad &\Longrightarrow \quad S_n = \overline{S_n} + 2^{32} \geq \textsf{min32} + 2^{32} = 2^{31} > \textsf{max32} \geq max_n \\ (7) \quad &\Longrightarrow \quad S_n = \overline{S_n} - 2^{32} \leq \textsf{max32} - 2^{32} = -2^{31} - 1 < \textsf{min32} \leq min_n \end{aligned}$$

This yields Program 7, that also works in the $min_n = max_n$ case. □

Lemma 2 *If $min_n \neq max_n$ then we have:*

$$max_n - \textsf{max32} \leq \overline{S_n} - s \leq min_n - \textsf{min32} \quad \Longleftrightarrow \quad S_n = \overline{S_n} \tag{5}$$

$$\overline{S_n} - s < max_n - \textsf{max32} \quad \Longleftrightarrow \quad S_n = \overline{S_n} + 2^{32} \tag{6}$$

$$\overline{S_n} - s > min_n - \textsf{min32} \quad \Longleftrightarrow \quad S_n = \overline{S_n} - 2^{32} \tag{7}$$

$u_0 := s$
$v_0 := 0$
$min_0 := \mathsf{min32}$
$max_0 := \mathsf{max32}$
for $i = 0$ to $\frac{n}{2} - 1$
  $u_{i+1} := \mathsf{saturate}(u_i + a[i])$
  $v_{i+1} := v_i \overset{32}{+} a[i + \frac{n}{2}]$
  $min_{i+1} := \mathsf{saturate}(min_i + a[i + \frac{n}{2}])$
  $max_{i+1} := \mathsf{saturate}(max_i + a[i + \frac{n}{2}])$
end for
$inf := max_{\frac{n}{2}} - \mathsf{max32}$
$sup := min_{\frac{n}{2}} - \mathsf{min32}$
if $inf \le v_{\frac{n}{2}} \le sup$ then
  return $\mathsf{clip}_{min_{\frac{n}{2}}}^{max_{\frac{n}{2}}}(u_{\frac{n}{2}} + v_{\frac{n}{2}})$
end if
return max_n if $v_{\frac{n}{2}} < inf$
return min_n if $v_{\frac{n}{2}} > sup$

Program 5

```
long inf, sup;
int u = s, v = 0;
int min = INT_MIN, max = INT_MAX;
for (i = 0; i < n/2; i++) {
  u = L_mac(u, x[i], y[i]);
  v += L_mult(x[i+n/2], y[i+n/2]);
  min = L_mac(min, x[i+n/2], y[i+n/2]);
  max = L_mac(max, x[i+n/2], y[i+n/2]);
}
inf = max - INT_MAX;
sup = min - INT_MIN;
if (inf <= v && v <= sup) {
  if ((long)u + v > max) return max;
  if ((long)u + v < min) return min;
  return u + v;
}
if (v < inf) return max;
return min;
```

Program 6

Figure 7: Saturated reductions: parallelized code with only 32-bit accumulations.

If $a[i] \ge 0$, $max_{i+1} = \mathsf{saturate}(max_i + a[i]) \le max_i + a[i]$ by Lemma 1, so $max_{i+1} - max_i \le a[i] = S_{i+1} - S_i$. Else if $a[i] < 0$, $max_{i+1} = \mathsf{saturate}(max_i + a[i]) = max_i + a[i]$, and $max_{i+1} - max_i = a[i] = S_{i+1} - S_i$. Otherwise max_{i+1} has reached $\mathsf{min32}$, a contradiction since $\mathsf{min32} \le min_{i+1} \le max_{i+1} = \mathsf{min32} \Rightarrow min_{i+1} = max_{i+1} \Rightarrow min_n = max_n$. Summing the n inequalities $max_{i+1} - max_i \le S_{i+1} - S_i$ yields $max_n - max_0 \le S_n - S_0 \Leftrightarrow max_n - \mathsf{max32} \le S_n - s$. Similarly, $S_n - s \le min_n - \mathsf{min32}$. This yields (8):

$$max_n - \mathsf{max32} \le \quad S_n - s \quad \le min_n - \mathsf{min32} \tag{8}$$
$$-2^{32} + 1 = \mathsf{min32} - \mathsf{max32} < \quad S_n - s \quad < \mathsf{max32} - \mathsf{min32} = 2^{32} - 1 \tag{9}$$
$$-2^{32} + 1 = \mathsf{min32} - \mathsf{max32} \le \quad \overline{S_n} - s \quad \le \mathsf{max32} - \mathsf{min32} = 2^{32} - 1 \tag{10}$$

Equation (9) is implied by (8), while (10) holds because $\overline{S_n}$ and $s \in [\mathsf{min32}, \mathsf{max32}]$. Since S_n and $\overline{S_n}$ are computed the same way, except for the modular 32-bit addition in case of $\overline{S_n}$, we are left with only three possibilities:

- $S_n = \overline{S_n}$ Then (8) reduces to case (5).
- $S_n = \overline{S_n} + 2^{32}$ We show by contradiction that $\overline{S_n} - s < max_n - \mathsf{max32}$, thus reducing to case (6). From (8), we have $\overline{S_n} - s + 2^{32} \le min_n - \mathsf{min32}$. Subtracting those inequalities yields $2^{32} \le min_n - max_n + \mathsf{max32} - \mathsf{min32} \Leftrightarrow max_n \le min_n + \mathsf{max32} - \mathsf{min32} - 2^{32} = min_n - 1 < min_n$.
- $S_n = \overline{S_n} - 2^{32}$ Likewise one can show by contradiction that $\overline{S_n} - s > min_n - \mathsf{min32}$, thus reducing to case (7).

Finally, as min_n and max_n are signed 32-bit integers, $max_n - min_n \le \mathsf{max32} - \mathsf{min32} \Leftrightarrow max_n - \mathsf{max32} \le min_n - \mathsf{min32}$. Therefore, the inequalities $\overline{S_n} - s < max_n - \mathsf{max32}$ and $\overline{S_n} - s > min_n - \mathsf{min32}$ are exclusive.
□

Corollary 2 *Program 5 computes the same result as Program 3.*

Proof identical to the proof of Corollary 1. □

$S_0 := s$
$min_0 :=$ min32
$max_0 :=$ max32
for $i = 0$ to $n-1$
    $S_{i+1} := S_i + a[i]$
    $min_{i+1} :=$ saturate$(min_i + a[i])$
    $max_{i+1} :=$ saturate$(max_i + a[i])$
end for
return $\text{clip}_{min_n}^{max_n}(S_n)$

Program 4'

$\overline{S_0} := s$
$min_0 :=$ min32
$max_0 :=$ max32
for $i = 0$ to $n-1$
    $\overline{S_{i+1}} := \overline{S_i} \overset{32}{+} a[i]$
    $min_{i+1} :=$ saturate$(min_i + a[i])$
    $max_{i+1} :=$ saturate$(max_i + a[i])$
end for
$inf := max_n -$ max32
$sup := min_n -$ min32
return $\text{clip}_{min_n}^{max_n}(\overline{S_n})$ if $inf \le \overline{S_n} - s \le sup$
return max_n if $\overline{S_n} - s < inf$
return min_n if $\overline{S_n} - s > sup$

Program 7

Figure 8: Saturated reductions: use of 32-bit modulo / saturated accumulations.

3 Approximate parallel reductions

3.1 Problem Statement and Notations

Section 2 illustrates that satisfying the "bit-exact" requirements when unroll-and-jam does not apply wastes computational resources: four parallel MACs are required in order to run twice as fast as the original saturated reduction. In this section, we discuss several "approximate" transformations of the saturated reductions, that are suited to DSPs fitted with only two parallel MACs. All these transformations run twice as fast as the original saturated reduction, but more or less approximate the "bit-exact" result.

The approximate parallel saturated reductions discussed are:

$$\begin{aligned}
S_1 &\overset{\text{def}}{=} \lambda s.\text{saturate}\left(\bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[i]) + \textstyle\sum_{i=\frac{n}{2}}^{n-1} a[i]\right) \\
S_2 &\overset{\text{def}}{=} \lambda s.\text{saturate}\left(\bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[i]) + \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])\right) \\
S_3 &\overset{\text{def}}{=} \lambda s.\text{saturate}\left(s + \textstyle\sum_{i=0}^{\frac{n}{2}-1} a[2i] + \sum_{i=0}^{\frac{n}{2}-1} a[2i+1]\right) \\
S_4 &\overset{\text{def}}{=} \lambda s.\text{saturate}\left(\bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[2i]) + \bigoplus_{i=0}^{\frac{n}{2}-1}(a[2i+1])\right)
\end{aligned}$$

The common theme of these approximate algorithms is to split the reduction into sub-reductions, which are computed in parallel and combined at the end using a saturated addition. When using non-saturated arithmetic (modular integer arithmetic), overflows must be avoided by using wider precision arithmetic, such as 40-bit integers on the new DSP-MCUs. Another difference between approximate algorithms is that some of them expose spatial locality of memory accesses, such as S_3 and S_4.

The correctness of an approximate algorithm mainly depends on the potential saturation of the sub-reductions. Let us introduce the notations:

$$\underset{i=m}{\overset{n}{\text{MAX}}}\Sigma\, a[i] \overset{\text{def}}{=} \max_{m \le j \le n}\left[\sum_{k=m}^{j} a[k]\right] \qquad \underset{i=m}{\overset{n}{\text{MIN}}}\Sigma\, a[i] \overset{\text{def}}{=} \min_{m \le j \le n}\left[\sum_{k=m}^{j} a[k]\right]$$

Then the different approximation cases we shall discuss for $i \in [j, k[$ are:

case 1 No saturation: $\text{min32} \le \text{MIN}\Sigma_{i=j}^{k-1} a[i] \wedge \text{MAX}\Sigma_{i=j}^{k-1} a[i] \le \text{max32}$.

case 2 Saturation on one side: $\text{min32} \le \text{MIN}\Sigma_{i=j}^{k-1} a[i] \wedge \text{max32} < \text{MAX}\Sigma_{i=j}^{k-1} a[i]$, or $\text{min32} > \text{MIN}\Sigma_{i=j}^{k-1} a[i] \wedge \text{MAX}\Sigma_{i=j}^{k-1} a[i] \le \text{max32}$.

case 3 Saturation on both sides.

Each approximate algorithm achieves a tradeoff between three points:

Accuracy The main reason for using saturated arithmetic in fixed-point digital signal processing, instead of integer modular arithmetic, is the better behavior of the former in linear filtering applications. However, as saturations in linear filters implies signal distortion, it is a design goal of fixed-point digital signal processing algorithms to avoid saturation as much as possible. Consequently, we sort those three different cases in order of importance:

$$\text{case } 1 > \text{case } 2 > \text{case } 3$$

Interleaving There are two basic ways to split the reduction:

- either separate odd and even index, as $\sum_{i=0}^{n-1} a[i] = \sum_{i=0}^{\frac{n}{2}-1} a[2i] + \sum_{i=0}^{\frac{n}{2}-1} a[2i+1]$.
- or maintain the main order by summing the first $\frac{n}{2}$ as well as the last $\frac{n}{2}$ elements together, $\sum_{i=0}^{n-1} a[i] = \sum_{i=0}^{\frac{n}{2}-1} a[i] + \sum_{i=\frac{n}{2}}^{n-1} a[i]$

Interleaving potentially yields better performances, as it exposes spatial locality between $a[2i]$ and $a[2i+1]$. On the new DSP-MCUs, spatial locality enables memory access packing, that is, loading or storing a pair of 16-bit numbers as a single 32-bit memory access.

32-bit versus 40-bit We will see that saturating the sub-sums usually gives worse results than summing in higher-precision arithmetic, then saturating in the end.

3.2 Main Approximation Results

In this section, we denote: $S \stackrel{\text{def}}{=} \lambda x. \bigoplus_{i=\frac{n}{2}}^{n-1}(x, a[i])$.

Let us denote by s_m the result of $\bigoplus_{i=0}^{m-1}(s, a[i])$, so that by $s_{\frac{n}{2}} = \bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[i])$. The original saturated reduction $s_n = \bigoplus_{i=0}^{n-1}(s, a[i]) = \bigoplus_{i=\frac{n}{2}}^{n-1}(s_{\frac{n}{2}}, a[i])$, and we are interested to know when $\lambda x. \bigoplus_{i=\frac{n}{2}}^{n-1}(x, a[i])$ saturates.

Lemma 3 (Saturation of S)

$$S \textit{ max-saturates} \iff x + \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] > \mathsf{max32} \tag{11}$$

$$S \textit{ min-saturates} \iff x + \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MIN}\Sigma}}\, a[i] < \mathsf{min32} \tag{12}$$

$$S \textit{ min-max-saturates} \implies \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] - \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MIN}\Sigma}}\, a[i] > \mathsf{max32} - \mathsf{min32} \tag{13}$$

$$S \textit{ does not saturate} \implies \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] - \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MIN}\Sigma}}\, a[i] \leq \mathsf{max32} - \mathsf{min32} \tag{14}$$

Relations (11) and (12) are straightforward from the definition of $\mathsf{MAX}\Sigma$ and $\mathsf{MIN}\Sigma$: because min and max are linear functions, x can be subtracted to each sum and moved out from the min and max. Relations (13) and (14) are obtained by subtractions on (11) and (12). □

Lemma 4 (Saturation and extrema) *Let us denote by* $S_m = \bigoplus_{i=0}^{m-1}(0, a[i])$*:*

If S_m max-saturates and does not min-saturate, then

$$S_m = \mathsf{max32} \iff \sum_{i=0}^{m-1} a[i] = \underset{i=0}{\overset{m-1}{\mathsf{MAX}\Sigma}}\, a[i]$$

If S_m min-saturates and does not max-saturate, then

$$S_m = \mathsf{min32} \iff \sum_{i=0}^{m-1} a[i] = \mathsf{MIN}\Sigma_{i=0}^{m-1} a[i]$$

Consider that $S_m = \mathsf{max32}$ does not saturate on min. Suppose, by contradiction, that there exists $j < m$ such that $\sum_{i=0}^{j-1} a[i] > \sum_{i=0}^{m-1} a[i]$. Then, we have $\sum_{i=j}^{m-1} a[i] < 0$. Eventually because $\bigoplus_{i=0}^{m-1}(a[i])$ does not saturate on min,

$$\begin{aligned}
\bigoplus_{i=j}^{m-1}\left(\bigoplus_{i=0}^{j-1}(a[i]), a[i]\right) &\leq \bigoplus_{i=0}^{j-1}(a[i]) + \sum_{i=j}^{m-1} a[i] \\
&< \bigoplus_{i=0}^{j-1}(a[i]) \\
&< \mathsf{max32}
\end{aligned}$$

Reciprocally, suppose $\sum_{i=0}^{m-1} a[i] = \mathsf{MAX}\Sigma_{i=0}^{m-1} a[i]$ and consider the largest index $0 \leq j \leq m$ such that $S_j = \mathsf{max32}$. We have $S_m = S_j + \sum_{i=j}^{m-1} a[i]$. Then, because $\sum_{i=j}^{m-1} a[i] \geq 0$, $S_m = \mathsf{max32}$. □

Theorem 3 (Exact value of S)

$$S \text{ does not saturate} \implies S(x) = x + \sum_{i=\frac{n}{2}}^{n-1} a[i] \tag{15}$$

$$S \begin{array}{l}\text{max-saturates}\\ \text{does not min-saturate}\end{array} \implies S(x) = \sum_{i=\frac{n}{2}}^{n-1} a[i] - \left(\mathsf{MAX}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] - \mathsf{max32}\right) \tag{16}$$

$$S \begin{array}{l}\text{min-saturates}\\ \text{does not max-saturate}\end{array} \implies S(x) = \sum_{i=\frac{n}{2}}^{n-1} a[i] - \left(\mathsf{MIN}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] - \mathsf{min32}\right) \tag{17}$$

$$S \text{ min-max-saturates} \implies S(x) = \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i]) \ \forall x \in [\mathsf{min32}, \mathsf{max32}] \tag{18}$$

⋄ To prove Relation (16) let us prove by induction on $m \geq j$ that if S_m saturates on max, not on min:

$$\left(S_m = \bigoplus_{i=0}^{m-1}(a[i])\right) = \sum_{i=0}^{m-1} a[i] - \left(\mathsf{MAX}\Sigma_{i=0}^{m-1} a[i] - \mathsf{max32}\right) \tag{19}$$

where j is the smallest index such that $\sum_{i=0}^{j-1} a[i] > \mathsf{max32}$.
Because $S_j = \mathsf{max32}$ and $\sum_{i=0}^{j-1} a[i] = \mathsf{MAX}\Sigma_{i=0}^{j-1} a[i]$, and the induction hypothesis is true for $m = j$. Let us consider m such that (19) is true. There are two possibilities:

1. $S_{m+1} = \mathsf{max32}$, and the result directly extends from Lemma 4.
2. $S_{m+1} = S_m + a[m] < \mathsf{max32}$, and thanks to Lemma 4, $\sum_{i=0}^{m} a[i] \neq \mathsf{MAX}\Sigma_{i=0}^{m} a[i]$, so $\mathsf{MAX}\Sigma_{i=0}^{m} a[i] = \mathsf{MAX}\Sigma_{i=0}^{m-1} a[i]$. The result follows from the relation on m.

⋄ Let us prove Relation (18). Suppose without loss of generality that $x \leq x'$. Because

$$x \leq y \implies \mathsf{saturate}(x + a) \leq \mathsf{saturate}(y + a)$$

We have,

$$\forall m, \bigoplus_{i=\frac{n}{2}}^{m-1}(x, a[i]) \leq \bigoplus_{i=\frac{n}{2}}^{m-1}(x', a[i]) \leq \mathsf{max32}$$

Now, since S saturates on max, if we denote by j an index where it saturates, we have

$$\mathsf{max32} = \bigoplus_{i=\frac{n}{2}}^{j-1}(x, a[i]) = \bigoplus_{i=\frac{n}{2}}^{j-1}(x', a[i])$$

$\square$

Corollary 3 (Comparison on case 1) *If S does not saturate, then:*

$$S(x) = \mathsf{saturate}\left(x + \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])\right) \iff \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i]) \text{ does not saturate}$$

The reciprocal is clearly true.
Suppose by contradiction that $S' = \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])$ saturates. Since S does not saturates, from relations (13) and (14), we can prove by contradiction that S' does not saturate both on max and min. Hence, consider without loss of generality that S' saturates on max, not on min. From Relation (16) we get

$$\begin{aligned}
\mathsf{saturate}\left(x + \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])\right) &= \mathsf{saturate}\left(x + \sum_{i=\frac{n}{2}}^{n-1} a[i] - \left(\underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] - \mathsf{max32}\right)\right) \\
&= \mathsf{saturate}\left(S - \left(\underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] - \mathsf{max32}\right)\right) \\
&= S - \left(\underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] - \mathsf{max32}\right) \\
&\neq S
\end{aligned}$$

$\square$

As a consequence of Theorem 3 and Corollary 3, in case 1 on $[\frac{n}{2}, n-1[$, S_1 is better than S_2.

Theorem 4 (Comparison on case 2) *Suppose $x \neq 0$, S max-saturates, and S does not min-saturate. Then:*

$$S(x) = \mathsf{saturate}\left(x + \sum_{i=\frac{n}{2}}^{n-1} a[i]\right) \iff \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i] \tag{20}$$

$$S(x) = \mathsf{saturate}\left(x + \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])\right) \iff \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i] \bigwedge x > 0 \tag{21}$$

Which is true in particular when $a[i] \geq 0\, \forall i$.

$\diamond$ Let us start with the proof of (20). From (16) we have

$$S(x) = x + \sum_{i=\frac{n}{2}}^{n-1} a[i] - \left(x + \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] - \mathsf{max32}\right)$$

Also from (11) we get that

$$x + \underset{i=\frac{n}{2}}{\overset{n-1}{\mathsf{MAX}\Sigma}}\, a[i] - \mathsf{max32} > 0$$

Consequently,

$$S(x) \neq x + \sum_{i=\frac{n}{2}}^{n-1} a[i]$$

and

$$S(x) = \mathsf{saturate}\left(x + \sum_{i=\frac{n}{2}}^{n-1} a[i]\right) \iff S(x) = \mathsf{max32}$$

Eventually, we apply Lemma 4.

⋄ To prove (21) we need to consider two cases:

1. $\bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])$ saturates on max. By applying Theorem 3 both on $\bigoplus_{i=\frac{n}{2}}^{n-1}(x, a[i])$ and $\bigoplus_{i=\frac{n}{2}}^{n-1}(0, a[i])$, we get $S(x) = \bigoplus_{i=\frac{n}{2}}^{n-1}(0, a[i])$. Now, $x \neq 0$ and $x + \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i]) \neq S(x)$. So, $\mathsf{saturate}\left(x + \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])\right) = S(x)$ implies $S(x) = \mathsf{max32}$, and the equivalence relation follows.

2. $\bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])$ does not saturate on max. First, because S saturates on max, necessary $x > 0$. Moreover, $x + \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i]) \geq x + \sum_{i=\frac{n}{2}}^{n-1} a[i] > \bigoplus_{i=\frac{n}{2}}^{n-1}(x, a[i]) = S(x)$. Hence, $\mathsf{saturate}\left(x + \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])\right) = S(x) \iff S(x) = \mathsf{max32}$.

□

As a consequence of Theorem 4, in case 2 on $[\frac{n}{2}, n-1[$, S_1 is better than S_2.

Corollary 4 (Comparison on case 3) *If $x \neq 0$ and S min-max-saturates:*

$$S(x) = \mathsf{saturate}\left(x + \sum_{i=\frac{n}{2}}^{n-1} a[i]\right) \iff \bigvee \begin{cases} \mathsf{MAX}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i] \\ \mathsf{MIN}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i] \end{cases}$$

$$S(x) = \mathsf{saturate}\left(x + \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])\right) \iff \bigvee \begin{cases} \mathsf{MAX}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i] \bigwedge x > 0 \\ \mathsf{MIN}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i] \bigwedge x < 0 \end{cases}$$

Thus in case 3 on $[\frac{n}{2}, n-1[$, S_1 is better than S_2.

3.3 Classification of the Approximate Algorithms

On the light of previous remarks and theorems, we compared the four different algorithms (S_1, S_2, S_3, S_4) and summarized their characteristics: S_4 is the worst approximation because whenever one of its sub-reductions or the original sum $\bigoplus_{i=0}^{n-1}(s, a[i])$ saturates, the result is not "bit-exact".

The main drawback of S_1 is that it requires the target processor to operate on numbers larger than 32-bit, for instance 40-bit on most fixed-point DSPs. In that case, the loop iteration count has to be no larger than 2^8, else strip-mining of the reduction loop is required.

When the target processor is limited to 32-bit arithmetic, or suffers significant performance degradations when operating at higher precision, the S_2 or S_4 approximate techniques should be used to parallelize the saturated reductions.

	Condition for Correctness	Properties
S_1	⋄ case 1 on $[\frac{n}{2}, n[$ or $\mathsf{MAX}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i]$ or $\mathsf{MIN}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i]$	Needs 40-bit operations. Better than S_2 and S_3.
S_2	⋄ case 1 on $[\frac{n}{2}, n[\bigwedge$ no saturation of $\bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])$ or $\mathsf{MAX}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i] \bigwedge \bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[i]) > 0$ or $\mathsf{MIN}\Sigma_{i=\frac{n}{2}}^{n-1} a[i] = \sum_{i=\frac{n}{2}}^{n-1} a[i] \bigwedge \bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[i]) < 0$	Only 32-bit operations. Better than S_4.
S_3	⋄ case 1 on $[0, \frac{n}{2}[\bigwedge$ same conditions as S_1.	Spatial locality. Needs 40-bit operations. Better than S_4.
S_4	⋄ case 1 on $[0, n[$ and neither $\bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[2i])$ nor $\bigoplus_{i=0}^{\frac{n}{2}-1}(a[2i+1])$ saturates	Spatial locality. Only 32-bit operations. Simplest implementation. Worst approximation.

Also, so as to illustrate the non-correctness of several algorithms, we provide counter-examples in Table 1. In this table, we use the previously introduced notations, in addition to:

$$
\begin{aligned}
S_5 &\stackrel{\text{def}}{=} \lambda s.\mathsf{saturate}\left(s + \bigoplus_{i=1}^{n}(a[n-i])\right) \\
S_6 &\stackrel{\text{def}}{=} \lambda s.\mathsf{saturate}\left(\bigoplus_{i=0}^{\frac{n}{2}-1}(s, a[i]) + \bigoplus_{i=1}^{\frac{n}{2}}(a[n-i])\right) \\
S_7 &\stackrel{\text{def}}{=} \lambda s. \bigoplus_{i=\frac{n}{2}}^{n-1}(a[i])
\end{aligned}
$$

a										S	S_1	S_2	S_3	S_4	S_5	S_6	S_7
-1	7	5	-2	-4	1	10	7	2	-2	**13**	15	15	15	15	14	15	**13**
10	7	-8	1	-12	-1	13	4	-2	-7	**3**	**3**	2	5	5	5	**3**	6
6	6	5	-2	-10	-1	-5	13	4	-1	**13**	**13**	**13**	15	14	14	12	10
6	7	4	-2	3	-5	-4	2	3	-1	**10**	**10**	**10**	13	13	13	**10**	-5
-2	9	-13	3	-6	-2	5	7	1	8	**10**	**10**	6	**10**	5	4	4	15
1	2	-1	3	4	-3	1	-2	5	3	**13**	**13**	**13**	**13**	**13**	**13**	**13**	4

Table 1: Counter examples: for the sake of simplicity, the examples are scaled to $\mathsf{min32} = -16$, $\mathsf{max32} = 15$, $n = 10$, and $s = 0$.

4 Conclusions

This paper addresses the problem of improving the performance of the saturated reductions on fixed-point Digital Signal Processors (DSPs), under the requirement to implement "bit-exact" variants of the reference telecommunications algorithms such as the ETSI EFR-5.1.0, the ETSI AMR-NB, the ITU G.723.1, and the ITU G.729. This problem is motivated by the need to exploit the instruction-level parallelism available on

the new generation of DSP–MCUs, in particular the Texas Instruments C6200 and C6400 series [8], the StarCore SC140 [6], and the STMicroelectronics ST120 [7].

On the ETSI ans the ITU reference implementations, several saturated reductions loops can be parallelized by applying unroll-and-jam, that is, unrolling of the outer loop into the inner loop so as to create more parallel work. When unroll-and-jam is not applicable, the arithmetic properties of the saturation operator allow to compute two saturated reduction steps per cycle, at the expense of four multiply-accumulate operations per cycle. Our main technique requires one 40-bit integer accumulation, and three 32-bit saturated accumulations, per cycle. Then we show how to replace this 40-bit integer accumulation by a 32-bit integer accumulation.

When the "bit-exact" requirement can be relaxed, more efficient but approximate techniques can be used to parallelize the saturated reductions. Based on further arithmetic properties of the saturation operator, we compare the approximate techniques to each other. In particular, the commonly used technique S_4 that computes two interleaved saturated sub-reductions achieves the worst approximation. We find that the most precise approximate technique S_1 computes the first half of the reduction in order using 32-bit saturation, and the second half using 40-bit integer arithmetic.

References

[1] S. Carr, Y. Guan: *Unroll-and-Jam Using Uniformly Generated Sets* Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-97), pp. 349–357, Dec. 1997

[2] B. Dupont de Dinechin, F. de Ferrière, C. Guillon, A. Stoutchinin: *Code Generator Optimizations for the ST120 DSP-MCU Core* International Conference on Compilers, Architectures, and Synthesis for Embedded Systems – CASES, Nov. 2000.

[3] B. Dupont de Dinechin, C. Monat, P. Blouet, C. Bertin: *DSP-MCU Processor Optimization for Portable Applications* Microelectronic Engineering, Elsevier, vol. 54, no 1–2, Dec. 2000.

[4] European Telecommunications Standards Institute – ETSI: *GSM Technical Activity, SMG11 (Speech) Working Group*, http://www.etsi.org.

[5] International Telecommunication Union – ITU: http://www.itu.int.

[6] StarCore: *SC140 DSP Core Reference Manual* MNSC140CORE/D, Dec. 1999.

[7] STMicroelectronics: *ST120 DSP-MCU CORE Reference Guide* http://www.st.com/stonline/st100/.

[8] Texas Instruments: *TMS320C6000 CPU and Instruction Set Reference Guide* SPRU189E, Jan 2000.

[9] N. Yadav, J. Glossner, M. Schulte: *Parallel Saturating Fractional Arithmetic Units* Proceedings of the 9th Great Lakes Symposium on VLSI, pp. 214–217, Mar. 1999.