On the Efficiency of Interval Multiplication Algorithms

Evgenija D. Popova *

Abstract

In this paper we present the theoretical base for some modifications in interval multiplication algorithms. A diversity of proposed implementation approaches is summarized along with a discussion on their cost-efficiency. It is shown that some improvements can be achieved by utilizing some properties of interval multiplication formulae and no special hardware support. Both conventional and extended interval multiplication operations are considered.

Keywords : interval multiplication operation, performance analysis

1 Introduction

Recent years are characterized by an increasing number of interval arithmetic applications in science and industry, and a wide variety of interval arithmetic software tools. The necessity of interval software performance improvements for numerical intensive applications has led to a number of hardware designs involving interval arithmetic [12]. Recent efforts in establishing several interval conventions and standards facilitate the development of commercial hardware and compiler support for interval arithmetic. However, speed remains the most important goal for system suppliers. Providing high-speed support for interval arithmetic operations requires joint efforts of mathematicians and engineers.

In this paper we consider probably the hardest and most expensive interval operation — multiplication — from both theoretic and algorithmic point of view. Next Section presents a new formula for interval multiplication which gives an analytic proof and theoretical reasons for some improvements in the algorithms for interval multiplication. The algebraic extension of conventional interval arithmetic is considered as a general case retaining all properties of classical interval analysis and containing conventional interval arithmetic as a special case. All theoretical derivations will be done in the general case of extended intervals, while the implementation algorithms will be discussed

*Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Acad. G. Bonchev str., block 8, BG-1113 Sofia, Bulgaria, E-mail : epopova@bio.bas.bg
partially supported by the Bulgarian National Science Fund under grant No. I-507/95

for classical and extended multiplication operations separately. In Section 3 we discuss the efficiency of several algorithms implementing classical interval multiplication operation. Implementation of the outwardly rounded operation, based on floating-point arithmetic, is considered. Since changing the rounding direction is an expensive operation on many systems, we consider implementation algorithms using rounding in only one direction. This is possible because IEEE floating-point arithmetic [2] in round-to-negative-infinity mode is symmetric with the arithmetic in round-to-positive-infinity mode. The reader is supposed to be familiar with the definition of computer interval arithmetic [8]. On the example of the classical algorithm, based on nine sign-dependent cases, we demonstrate how some properties of interval multiplication formulae can be exploited to gain an increased efficiency of interval multiplication operation. Further on, we give a short overview of several algorithms representing different approaches for the implementation of interval multiplication operation and discuss their cost-effectiveness. Algorithms for interval multiplication in the extended interval spaces are presented in Section 4 and their implementation cost is compared to the implementation cost of algorithms for conventional interval multiplication.

2 End-Point Representation of Interval Multiplication

Consider the set $\mathcal{D} = \mathbb{IR} \cup \overline{\mathbb{IR}}$ of usual (proper) intervals $\mathbb{IR} = \{[a^-, a^+] \mid a^- \leq a^+, a^-, a^+ \in \mathbb{R}\}$ and improper intervals $\overline{\mathbb{IR}} = \{[a^-, a^+] \mid a^- \geq a^+, a^-, a^+ \in \mathbb{R}\}$. For an extended (generalized) interval $A = [a^-, a^+] \in \mathcal{D}$, $a^\lambda \in \mathbb{R}$ denotes the first or second end-point of A depending on the value of $\lambda \in \Lambda = \{+, -\}$. The binary variable λ is sometimes expressed as a "product" of two or more binary variables, $\lambda = \mu\nu$, $\mu, \nu \in \Lambda$, defined by $++ = -- = +$ and $+- = -+ = -$. Every interval $A \in \mathcal{D}$ can be characterized by several functionals which are essentially used in the subsequent discussion.

Direction $\tau : \mathcal{D} \to \Lambda$ is defined by

$$\tau(A) = \begin{cases} +, & \text{if } a^- \leq a^+; \\ -, & \text{if } a^- \geq a^+. \end{cases}$$

An extended interval A is called *proper*, if $\tau(A) = +$ and *improper* otherwise. Denote $\mathcal{T} = \{A \in \mathcal{D} \mid A = [0, 0] \text{ or } a^- a^+ < 0\}$.

For an interval $A \in \mathcal{D} \setminus \mathcal{T}$ *sign* $\sigma : \mathcal{D} \setminus \mathcal{T} \to \Lambda$ is defined by

$$\sigma(A) = \begin{cases} +, & \text{if } a^{-\tau(A)} \geq 0; \\ -, & \text{if } a^{\tau(A)} \leq 0. \end{cases} \tag{1}$$

In particular, σ is well defined over $\mathbb{R} \setminus 0$.

The *symmetry* of an interval with respect to the point zero is represented by

$\chi : \mathcal{D} \to [-1, 1]$

$$\chi_A = \begin{cases} -1, & \text{if } A = [0,0] \\ a^{-\nu(A)}/a^{\nu(A)}, & \text{otherwise,} \end{cases}$$

where $\nu(A) = \{+, \text{ if } |a^+| = |a^-|;\ \sigma(|a^+| - |a^-|), \text{ otherwise}\}$. It is obvious that $a^{\nu(A)} = \{a^+, \text{ if } \mid a^+ \mid \geq \mid a^- \mid;\ a^-, \text{otherwise}\}$.

In what follows we consider only multiplication operation which has the following end-point representation:

$$A \times B = \begin{cases} [a^{-\sigma(B)}b^{-\sigma(A)},\ a^{\sigma(B)}b^{\sigma(A)}], & A, B \in \mathcal{D} \setminus \mathcal{T}; \\ [a^{\sigma(A)\tau(B)}b^{-\sigma(A)},\ a^{\sigma(A)\tau(B)}b^{\sigma(A)}], & A \in \mathcal{D} \setminus \mathcal{T}, B \in \mathcal{T}; \\ [a^{-\sigma(B)}b^{\tau(A)\sigma(B)},\ a^{\sigma(B)}b^{\tau(A)\sigma(B)}], & A \in \mathcal{T}, B \in \mathcal{D} \setminus \mathcal{T}; \\ [\min\{a^-b^+, a^+b^-\}, \max\{a^-b^-, a^+b^+\}], & A, B \in \mathcal{T}, \\ & \tau(A) = \tau(B) = +; \\ [\max\{a^-b^-, a^+b^+\}, \min\{a^-b^+, a^+b^-\}], & A, B \in \mathcal{T}, \\ & \tau(A) = \tau(B) = -; \\ 0, & A, B \in \mathcal{T}, \\ & \tau(A) \neq \tau(B). \end{cases} \tag{2}$$

The restriction of this formula to the conventional interval space $\mathbb{IR}$ results in the well-known classical interval arithmetic formula [1]. Details about generalized interval arithmetic can be found e.g. in [5], [10].

The occurrence of min and max functions at the end-points of the result on multiplication of two zero-involving intervals hampers not only analytical derivations in interval analysis but affects the performance of corresponding computer operation as well. Recently, an explicit representation for the end-points of the interval product has been found by the end-points of the zero involving arguments.

Theorem 2.1 *For $A, B \in \mathcal{T}$ such that $\tau(A) = \tau(B) = \tau$*

$$A \times B \quad = \quad \begin{cases} [a^{-\nu(B)\tau}b^{\nu(B)},\ a^{\nu(B)\tau}b^{\nu(B)}], & \text{if } \chi_A \leq \chi_B; \\ [a^{\nu(A)}b^{-\nu(A)\tau},\ a^{\nu(A)}b^{\nu(A)\tau}], & \text{if } \chi_A \geq \chi_B. \end{cases}$$

A complete proof of this Theorem can be found in [10]. In what follows, we discuss the impact of this formula for the computer implementation of interval multiplication operation.

Denote $\mathcal{Z} = \{[t^-, t^+] \in \mathbb{IR} \mid t^- \leq 0 \leq t^+\}$

Corollary 2.1 *For $A, B \in \mathcal{Z}$*

$$A \times B \quad = \quad \begin{cases} [a^{-\nu(B)}b^{\nu(B)},\ a^{\nu(B)}b^{\nu(B)}], & \text{if } \chi_A \leq \chi_B; \\ [a^{\nu(A)}b^{-\nu(A)},\ a^{\nu(A)}b^{\nu(A)}], & \text{if } \chi_A \geq \chi_B. \end{cases}$$

Corollary 2.2 *For $A, B \in \mathcal{Z}$*

$$A \times^{-} B \;=\; \begin{cases} [a^{\nu(B)}b^{-\nu(B)},\ a^{-\nu(B)}b^{-\nu(B)}], & \text{if } \chi_A \le \chi_B; \\ [a^{-\nu(A)}b^{\nu(A)},\ a^{-\nu(A)}b^{-\nu(A)}], & \text{if } \chi_A \ge \chi_B, \end{cases}$$

where $\times^{-}$ is inner (nonstandard) interval multiplication (the definitions of inner interval operations can be found e.g. in [5]).

Proof. For $A, B \in \mathcal{Z}$

$$A \times B = [\min\{a^{-}b^{+}, a^{+}b^{-}\},\ \max\{a^{-}b^{-}, a^{+}b^{+}\}]$$

and

$$A \times^{-} B = [\max\{a^{-}b^{+}, a^{+}b^{-}\},\ \min\{a^{-}b^{-}, a^{+}b^{+}\}],$$

The representations of $A \times B$ and $A \times^{-} B$ complements each other, which proves this corollary.

Proposition 2.1 *For $A, B \in \mathcal{T}$ such that $\tau(A) = \tau(B)$*

$$\chi(A) \le \chi(B) \quad \Longleftrightarrow \quad a^{-\nu(A)}b^{\nu(B)} \le^{\nu(A)\nu(B)} a^{\nu(A)}b^{-\nu(B)},$$

where $\le^{\lambda} = \{\le,\ \text{if } \lambda = +;\ \ge,\ \text{if } \lambda = -\}$ for $\lambda \in \Lambda$.

The proof follows from the definition of χ-functional and the property $\sigma(a^{\nu(A)}) = \nu(A)\tau(A)$ for $A \in \mathcal{T}$.

Next Corollary from Theorem 2.1 and Proposition 2.1 shows that interval multiplication requires only three (not four) real products in the worst case of arguments involving zero.

Corollary 2.3 *For $A, B \in \mathcal{Z}$*

$$A \times B = \begin{cases} [a^{-}b^{+}, a^{+}b^{+}], \ \text{if } a^{-}b^{+} \le a^{+}b^{-} \\ [a^{+}b^{-}, a^{+}b^{+}], \ \text{if } a^{+}b^{-} \le a^{-}b^{+} \end{cases} \text{and} \quad \nu(A) = \nu(B) = +; \\ \begin{cases} [a^{+}b^{-}, a^{-}b^{-}], \ \text{if } a^{+}b^{+} \le a^{-}b^{-} \\ [a^{+}b^{-}, a^{+}b^{+}], \ \text{if } a^{-}b^{-} \le a^{+}b^{+} \end{cases} \text{and} \quad \nu(A) = + = -\nu(B); \\ \begin{cases} [a^{-}b^{+}, a^{+}b^{+}], \ \text{if } a^{-}b^{+} \le a^{+}b^{-} \\ [a^{-}b^{+}, a^{-}b^{-}], \ \text{if } a^{+}b^{-} \le a^{-}b^{+} \end{cases} \text{and} \quad -\nu(A) = \nu(B) = +; \\ \begin{cases} [a^{+}b^{-}, a^{-}b^{-}], \ \text{if } a^{+}b^{-} \le a^{-}b^{+} \\ [a^{-}b^{+}, a^{-}b^{-}], \ \text{if } a^{-}b^{+} \le a^{+}b^{-} \end{cases} \text{and} \quad \nu(A) = \nu(B) = -. \end{cases}$$

The possibility for using only three floating-point multiplications in the worst case of multiplication of two zero involving normal intervals was first mentioned in [6]. Next analysis gives the theoretical basis for the algorithms considered in [4], [6] and Section 3.4.

For $A \in \mathcal{D}$ denote

$$\mu(A) = \begin{cases} \sigma(A), & \text{if } A \in \mathcal{D} \setminus \mathcal{T}; \\ \nu(A), & \text{if } A \in \mathcal{T} \end{cases} \quad \text{and for } \lambda \in \Lambda, \quad \lambda A = \begin{cases} A, & \text{if } \lambda = +; \\ (-1)A, & \text{if } \lambda = -. \end{cases}$$

Next Lemma follows from the associativity and commutativity of interval multiplication.

Lemma 2.1 *For $A, B \in \mathcal{D}$*

$$A \times B \quad = \quad (\mu(A)\mu(B))\,(\mu(A)A \times \mu(B)B)\,,$$

that is

$$A \times B \quad = \quad \begin{cases} U \times V, & \mu(A) = \mu(B); \\ (-1)(U \times V), & \mu(A) \neq \mu(B), \end{cases}$$

where $U = \mu(A)A$, $V = \mu(B)B$ are such that $\mu(U) = \mu(V) = +$.

Lemma 2.2 *For $A \in \mathcal{T}$*

$$-A \in \mathcal{T},\ \tau(-A) = \tau(A), \qquad \mu(-A) = -\,\mu(A), \qquad \chi(-A) = \chi(A).$$

These properties can be verified by a straightforward examination.

Proposition 2.2 *For $A, B \in \mathcal{D}$ and $\lambda = \mu(A)\mu(B)$*

$$A \times B \quad = \quad \lambda \begin{cases} [u^- v^-,\ u^+ v^+], & A, B \in \mathcal{D} \setminus \mathcal{T}; \\ [u^{\tau(B)} v^-,\ u^{\tau(B)} v^+], & A \in \mathcal{D} \setminus \mathcal{T},\ B \in \mathcal{T}; \\ [u^- v^{\tau(A)},\ u^+ v^{\tau(A)}], & A \in \mathcal{T},\ B \in \mathcal{D} \setminus \mathcal{T}; \\ [u^{-\tau} v^+,\ u^\tau v^+], & A, B \in \mathcal{T},\ \tau(A) = \tau(B) = \tau, \\ & \quad u^- v^+ \leq u^+ v^-; \\ [u^+ v^{-\tau},\ u^+ v^\tau], & A, B \in \mathcal{T},\ \tau(A) = \tau(B) = \tau, \\ & \quad u^+ v^- \leq u^- v^+; \\ 0, & A, B \in \mathcal{T}, \tau(A) \neq \tau(B), \end{cases}$$

wherein $[u^-, u^+] = \mu(A)A$, $[v^-, v^+] = \mu(B)B$.

The proof follows from Theorem 2.1, Proposition 2.1 and previous two Lemmas.

Corollary 2.4 *For $A, B \in \mathbb{IR}$ and $\lambda = \mu(A)\mu(B)$*

$$A \times B \quad = \quad \lambda \begin{cases} [u^- v^-,\ u^+ v^+], & A, B \in \mathbb{IR} \setminus \mathcal{Z}; \\ [u^+ v^-,\ u^+ v^+], & A \in \mathbb{IR} \setminus \mathcal{Z},\ B \in \mathcal{Z}; \\ [u^- v^+,\ u^+ v^+], & A \in \mathcal{Z},\ B \in \mathbb{IR} \setminus \mathcal{Z}; \\ [u^+ v^-,\ u^+ v^+], & A, B \in \mathcal{Z},\ u^+ v^- \leq u^- v^+; \\ [u^- v^+,\ u^+ v^+], & A, B \in \mathcal{Z},\ u^- v^+ \leq u^+ v^-. \end{cases}$$

wherein $[u^-, u^+] = \mu(A)A$, $[v^-, v^+] = \mu(B)B$.

3 Algorithms for Classical Interval Product

3.1 Sequential Algorithms

Usually, designers make choice between two classical alternatives for the implementation of interval multiplication. First alternative involves nine cases determined by the algebraic signs of the end-points of the interval operands (the restriction of formula (2) to $\mathbb{IR}$). Second alternative involves computation of minimum and maximum of the four possible products of the operands end-points. The average number of floating-point multiplications required for the first alternative is less than that required by the second one. Implemented in software, the relative efficiency of both alternatives are architecture dependent, although the first alternative is often preferred in low-level implementations designed for efficiency. As a typical representative of the first alternative we consider Algorithm 3.1, given in [3]. This algorithm computes the product $A \times B$ for $A, B \in \mathcal{Z}$ by finding maximum of four floating-point products and requires four floating-point multiplications and two floating-point comparison operations as a whole. According to Corollary 2.3, interval result in the considered case can be obtained only by three floating-point multiplications. Hence, Algorithm 3.1 can be modified at the step corresponding to $A, B \in \mathcal{Z}$, so that interval result be computed by three floating-point multiplications and three floating-point comparison operations (see Algorithm 3.2).

Remark 3.1 Note, that this modification does not require any change in the other concepts (e. g. for supported intervals or exception handling) adopted in [3].

3.2 Zero involving arguments

In this paper we consider the prevealing case of interval arithmetic implementations that, although being in IEEE floating-point environment [2], do not distinguish the sign of zero. All considerations in this section remain valid, with minor modifications, for intervals involving signed zero.

From set-theoretical point of view $0 \in [-a, 0]$ and $0 \in [0, a]$, where $a \in \mathbb{R}, a > 0$. However, from algebraic point of view the sign of the interval $A = [a^-, a^+] \in \mathcal{D}$ such that $a^- a^+ = 0$ and $a^- \neq a^+$, is well defined by the nonzero end-point sign (see (1)). It can be easily verified the correctness of the following

Proposition 3.1 *For $A = [a^-, a^+] \in \mathcal{D}$, with $a^- a^+ \geq 0$ and $B = [b^-, b^+] \in \mathcal{D}$, with $b^- b^+ = 0$, the interval product $A \times B$ is such that*

$$\left.\begin{array}{l} [a^{\sigma(A)\tau(B)}b^{-\sigma(A)},\ a^{\sigma(A)\tau(B)}b^{\sigma(A)}], \\ \qquad\qquad\qquad\qquad \textit{if } a^- a^+ > 0; \\ [\min\{a^-b^+, a^+b^-\},\ \max\{a^-b^-, a^+b^+\}], \\ \qquad \textit{if } a^- a^+ = 0,\ \tau(A) = \tau(B) = +; \\ [\max\{a^-b^-, a^+b^+\},\ \min\{a^-b^+, a^+b^-\}], \\ \qquad \textit{if } a^- a^+ = 0,\ \tau(A) = \tau(B) = - \end{array}\right\} = [a^{-\sigma(B)}b^{-\sigma(A)},\ a^{\sigma(B)}b^{\sigma(A)}].$$

Algorithm 3.1 *Interval multiplication according to [3].*
$A = [a^-, a^+],\ \ B = [b^-, b^+],\ \ A \times B = [r^-, r^+]$

```
Save_Rounding_Mode;
Set_Rounding_Mode_Up;
if a⁻ > 0   {A > 0}
  then  if b⁻ > 0 then  {B > 0}
                        r⁻ = −(−a⁻b⁻)
                        r⁺ = (a⁺b⁺)
                elseif  b⁺ < 0 then  {B < 0}
                                     r⁻ = −(−a⁺b⁻)
                                     r⁺ = (a⁻b⁺)
                                else   {0 ∈ B}
                                     r⁻ = −(−a⁺b⁻)
                                     r⁺ = (a⁺b⁺)
  elseif a⁺ < 0   {A < 0}
       then  if b⁻ > 0 then  {B > 0}
                             r⁻ = −(−a⁻b⁺)
                             r⁺ = (a⁺b⁻)
                        elseif b⁺ < 0 then  {B < 0}
                                            r⁻ = −(−a⁺b⁺)
                                            r⁺ = (a⁻b⁻)
                                       else   {0 ∈ B}
                                            r⁻ = −(−a⁻b⁺)
                                            r⁺ = (a⁻b⁻)
       elseif b⁻ > 0  {0 ∈ A}
            then  {B > 0}
                  r⁻ = −(−a⁻b⁺)
                  r⁺ = (a⁺b⁺)
            elseif b⁺ < 0
                 then  {B < 0}
                       r⁻ = −(−a⁺b⁻)
                       r⁺ = (a⁻b⁻)
                 else   {0 ∈ B}
                       r⁻ = − max(−a⁻b⁺, −a⁺b⁻)
                       r⁺ = max(a⁻b⁻, a⁺b⁺)
if isnan(r⁻) or isnan(r⁺)
  then if not (isnan(a⁻) or isnan(b⁻))  then r⁻ = −∞
                                             r⁺ = +∞

Restore_Rounding_Mode;
return [r⁻, r⁺]
```

Algorithm 3.2 *Improved algorithm for interval multiplication.*
$A = [a^-, a^+],\ \ B = [b^-, b^+],\ \ A \times B = [r^-, r^+]$

Save_Rounding_Mode;
Set_Rounding_Mode_Up;
if $a^- \geq 0 \quad \{A \geq 0\}$
  then if $b^- \geq 0$ then $\{B \geq 0\}$ $r^- = -(-a^- b^-)$
                                $r^+ = (a^+ b^+)$
       elseif $b^+ \leq 0$ then $\{B < 0\}$ $r^- = -(-a^+ b^-)$
                                $r^+ = (a^- b^+)$
       else $\{0 \overset{\circ}{\in} B\}$ $r^- = -(-a^+ b^-)$
                                $r^+ = (a^+ b^+)$
  elseif $a^+ \leq 0 \quad \{A < 0\}$
    then if $b^- \geq 0$ then $\{B \geq 0\}$ $r^- = -(-a^- b^+)$
                                $r^+ = (a^+ b^-)$
         elseif $b^+ \leq 0$ then $\{B < 0\}$ $r^- = -(-a^+ b^+)$
                                $r^+ = (a^- b^-)$
         else $\{0 \overset{\circ}{\in} B\}$ $r^- = -(-a^- b^+)$
                                $r^+ = (a^- b^-)$
    elseif $b^- \geq 0 \quad \{0 \overset{\circ}{\in} A\}$
      then $\{B \geq 0\}$ $r^- = -(-a^- b^+)$
                          $r^+ = (a^+ b^+)$
      elseif $b^+ \leq 0$
        then $\{B < 0\}$ $r^- = -(-a^+ b^-)$
                         $r^+ = (a^- b^-)$
        else $\{0 \overset{\circ}{\in} B\}$
        if $a^+ < -a^-$ then if $b^+ < -b^-$
                             then $r^- = -\max(-a^- b^+, -a^+ b^-)$
                                  $r^+ = a^- b^-$
                             else $r^- = -(-a^- b^+)$
                                  $r^+ = \max(a^- b^-, a^+ b^+)$
                        elseif $b^+ < -b^-$
                             then $r^- = -(-a^+ b^-)$
                                  $r^+ = \max(a^- b^-, a^+ b^+)$
                             else $r^- = -\max(-a^- b^+, -a^+ b^-)$
                                  $r^+ = (a^+ b^+)$
if isnan(r^-) or isnan(r^+)
  then if not (isnan(a^-) or isnan(b^-)) then $r^- = -\infty$
                                                  $r^+ = +\infty$
Restore_Rounding_Mode;
return $[r^-, r^+]$

This Proposition says that, when one or both arguments involve zero at one or both end-points, instead of using one of the formulae in left-hand side of the above equality, we can use the formula for two nonzero arguments. Let us consider now what are the consequences, imposed by an implementation, guided only by the set-theoretical properties of intervals, for the run-time performance of the algorithms.

Both approaches (set-theoretic and algebraic) result in algorithms having equal performance if only one of the arguments involves zero at the end-point(s). This is not the case when both arguments involve zero at the end-point(s). The Algorithm 3.1 processes operation $A \times B$, for arguments

$$\begin{array}{lll} A = [a^-, a^+], & a^- a^+ = 0, & a^- < a^+; \\ B = [b^-, b^+], & b^- b^+ = 0, & b^- < b^+, \end{array} \tag{3}$$

by the branch labeled $\{0 \in A\}, \{0 \in B\}$, while the improved Algorithm 3.2 processes the operation with same arguments by those branches corresponding to the signs of the arguments. Since the case $0 \in A$, $0 \in B$ is computationally the heaviest among the nine sign-dependent cases, the Algorithm 3.2 possesses better performance then the Algorithm 3.1 for arguments defined by (3). A comparison of both algorithms is presented in Table 1.

$\sigma(A)$ \ $\sigma(B)$	$+$	$-$
$+$	$2\,\triangleright,\ 2\bowtie,\ 2*$	$1\,\triangleright,\ 2\bowtie,\ 2*$
$-$	$1\,\triangleright,\ 2\bowtie,\ 2*$	$2\bowtie,\ 2*$

Table 1: Number of extra sign-tests ($\triangleright$), floating-point comparisons ($\bowtie$) and floating-point multiplication operations ($*$) performed by Algorithm 3.1 than the Algorithm 3.2 in computation of $[a^-, a^+] \times [b^-, b^+]$, where $a^- a^+ = b^- b^+ = 0$, and $a^- < a^+,\ b^- < b^+$.

Example 3.1 *Compute* $[-2, 0] \times [0, 3]$ *by Algorithm 3.1 and Algorithm 3.2.*

Algorithm 3.1 computes $[r^-, r^+] = [-\max\{6, -0\},\ \max\{-0, 0\}] = [-6, 0]$ by 4 sign-tests, 4 floating-point multiplications and 2 floating-point comparisons, while Algorithm 3.2 computes $[r^-, r^+] = [-6, 0]$ by 3 sign-tests and 2 floating-point multiplications.

3.3 Handling of NaNs

Algorithm 3.1 [3], designed for IEEE 754 compliant processors, provides a non-stop handling of Invalid Operation exception arising on floating-point operations

$0\times(\pm\infty)$. The IEEE 754 default exceptional result `qNaN` is used at the end of the Algorithm 3.1 (last two `if` statements) to detect when a result end-point needs to be set to $\pm\infty$ according to the exception handling mechanism, proposed in [3]. Handling interval arguments, both involving zero at their end-point(s), the same way as arguments, both containing zero in the interior, (case $0 \in A$, $0 \in B$) requires some additional waste of efficiency. The following example illustrates this effect.

Example 3.2 *Compute* $[-3, 0] \times [0, \infty]$ *by Algorithm 3.1 and Algorithm 3.2.*

Algorithm 3.1 computes:	*Algorithm 3.2 computes:*
$r^- = -\max\{-\infty, 0\}$	$r^- = -\infty$
$r^+ = \max\{0, \texttt{NaN}\}$	$r^+ = 0$

To handle `NaNs` correctly determining r^+, Algorithm 3.1 requires either some hardware primitives that support efficient implementation of max satisfying

$$\max(x, \texttt{NaN}) \quad = \quad \texttt{NaN} \qquad \text{for any } x,$$

or max, based on IEEE comparisons, should be implemented by one comparison more using the unordered paradigm [9]. The alternative ("algebraic") approach, used by Algorithm 3.2, provides efficiency in this situation, too, because no Invalid Operation exception $0 \times (\pm\infty)$ can arise in the branch $\{0 \mathbin{\mathring{\in}} A,\ 0 \mathbin{\mathring{\in}} B\}$, using max-function (i.e. floating-point comparison). This branch of Algorithm 3.2 handles only the product of two empty set intervals, represented by `[NaN, NaN]`, which does not cause problems because both arguments of the IEEE comparison in this case are `NaN`.

Remark 3.2 In addition to the efficiency, Algorithm 3.2 gains a sharper interval result on floating-point Invalid Operation exception. Latter concerns, however, the concept of exception handling which will not be discussed here.

Remark 3.3 We moved last two `if` statements from Algorithm 3.1 to Algorithm 3.2 without any change because we do not discuss the exception handling mechanism in this paper.

3.4 Parallel Approach

The Algorithms 3.1 and 3.2 are entirely sequential and thus both will be less than optimal on modern, heavily pipelined superscalar processors. The tests and branches in these algorithms tend to prevent hardware and compilers from cooperating to fully utilize the floating-point pipeline to exploit instruction-level parallelism. The idea for a brute force approach in implementation of interval multiplication operation was proposed in [7]. It was suggested that an implementation of interval multiplication be based on the usual definition for $A, B \in \mathbb{IR}$

$$A \times B \quad = \quad [\min\{a^-b^-, a^-b^+, a^+b^-, a^+b^+\},\ \max\{a^-b^-, a^-b^+, a^+b^-, a^+b^+\}]$$

and some hardware support for branchless implementation of min and max instructions, so that the result can be computed with eight multiplications and six min/max operations but no explicit conditions. "The general case could exploit dual multipliers if available in hardware, although a single pipelined multiplier that could initiate a new operation on every cycle might suffice."

Further discussion on this approach and the design of special hardware, dedicated for interval operations, is provided in [13]. Based on the assumption that one arithmetic operation as well as a floating-point comparison lasts one unit of time while switches controlled by the sign bit are free of charge, the Algorithm 3.3 was pointed out in [13] as optimal. If two floating-point multipliers and one floating-point comparison units are available in parallel, this algorithm performs interval multiplication in case of $0 \in A,\ 0 \in B$ in four time steps and all other cases of interval multiplication – in one time step. Applying the considerations from Sections 3.2 and 3.3, this algorithm and the others, considered in [13], will gain an increased performance for the special cases of arguments involving zero at the end-point(s). However, whether sign-tests can be neglected or not is system dependent. On some superscalar processors testing the floating-point sign bit is performed by using integer operations. Hence, the Algorithm 3.3 will not be optimal on such systems.

Algorithm 3.3 *Interval multiplication according to [13].*
$A = [a^-, a^+],\ B = [b^-, b^+],\ A \times B = R$
$\bigtriangledown$, $\bigtriangleup$ *denote floating-point product rounded toward* $-\infty$, *resp.* $+\infty$.

1. $s^- = (\text{ if } (b^- \geq 0 \vee (a^+ < 0 \wedge b^+ \geq 0))\ \text{ then }\ a^-\ \text{ else }\ a^+\ ) \bigtriangledown$
   $(\text{ if } (a^- \geq 0 \vee (a^+ \geq 0 \wedge b^+ < 0))\ \text{ then }\ b^-\ \text{ else }\ b^+\ )$
   $s^+ = (\text{ if } ((b^- \geq 0 \vee (a^+ \geq 0 \wedge b^+ \geq 0))\ \text{ then }\ a^+\ \text{ else }\ a^-\ ) \bigtriangleup$
   $(\text{ if } (a^- \geq 0 \vee (a^+ \geq 0 \wedge b^+ \geq 0))\ \text{ then }\ b^+\ \text{ else }\ b^-\ )$

   if $(a^- < 0 \wedge a^+ \geq 0 \wedge b^- < 0 \wedge b^+ \geq 0)$ then step (2), (3), (4)
   else $[r^-, r^+] = [s^-, s^+]$

2. $p = a^- \bigtriangledown b^+;\ \ q = a^+ \bigtriangledown b^-$

3. $r^- = \min(p, q);\ \ p = a^- \bigtriangleup b^-;\ \ q = a^+ \bigtriangleup b^+$

4. $r^+ = \max(p, q)$

A completely different approach for the implementation of interval multiplication was proposed in [4] (see also [1]) and modified in [6] to reduce the number of floating-point products in multiplication of two zero containing intervals from four to three. This approach uses an initial transformation of the arguments to

have non negative sign and ν-functional value, so that Corollary 2.4 be applied (see Algorithm 3.4). A sequential implementation of this approach is superior to the others for interval operands $A, B \in \mathcal{Z}$ but in all other sign-dependent cases this approach requires at least one floating-point comparison more than the Algorithm 3.2. Algorithm 3.4 presents the sequential algorithm from [6], slightly modified, to show its potential for parallelisation. Transformation of interval arguments could be performed in parallel. Upper bound (r^+) of the result could be computed in parallel to the computation of lower bound (r^-), too. Finding lower bound (r^-) of the result requires two sign-tests in order to avoid the necessity of special support for correct floating-point comparison with `NaNs` (see Section 3.2). This algorithm possesses considerably less conditionals than Algorithm 3.2 and equal performance for all sign-dependent cases of interval arguments. It supports all efficiency considerations from the previous sections and do not require special hardware support providing correct comparison with `NaNs`. We believe that the execution of this algorithm may be essentially accelerated by parallel processing and some pipeline techniques.

Algorithm 3.4 *Interval multiplication, parallel version of [6].*
$A = [a^-, a^+],\ \ B = [b^-, b^+],\ \ A \times B = R$

Save_Rounding_Mode;
Set_Rounding_Mode_Up;

1′. $t = -a^+$
$\mu A = t > a^-$
if μA then $a^+ = a^-$
$a^- = t$
else $a^- = -a^-$

1″. $t = -b^+$
$\mu B = t > b^-$
if μB then $b^+ = b^-$
$b^- = t$
else $b^- = -b^-$

2. if $b^- < 0$
then if $a^- < 0$ then $r^- = -(a^- b^+)$ $\|$ $t = -(a^+ b^-)$
$r^+ = a^+ b^+$ $\|$ if $t < r^-$ then $r^- = t$
else $r^+ = a^+ b^+$ $\|$ $r^- = -(a^+ b^-)$
else if $a^- < 0$ then $r^+ = a^+ b^+$ $\|$ $r^- = -(a^- b^+)$
else $r^+ = a^+ b^+$ $\|$ $r^- = -(-a^- b^-)$

3. Restore_Rounding_Mode
if $\mu A = \mu B$ then $R = [r^-, r^+]$
else $R = [-r^+, -r^-]$

4 Nonstandard Interval Products

It was demonstrated in [11], that a credible implementation of inner interval multiplication $\times^-$ requires three floating-point multiplications for each sign-dependent case. The algorithm, given in [11], can be modified to process the case $0 \overset{\circ}{\in} A$, $0 \overset{\circ}{\in} B$ according to Corollary 2.2. Conventional interval arithmetic, extended by inner interval operations, provides the same functionality as the algebraic extension, called here generalised interval arithmetic [5]. Former approach, however, requires implementation of eight interval operations, which is a considerable drawback.

An implementation of generalised interval arithmetic provides extended functionality [5] at a cost-efficiency compared to that of conventional interval arithmetic. For an efficient implementation, we propose Algorithm 4.1, wherein

$$\theta_A = \begin{cases} \text{true}, & \text{if } \mu(A) = -; \\ \text{false}, & \text{if } \mu(A) = + \end{cases} \quad \text{and} \quad \mu(A) = \begin{cases} \sigma(A), & \text{if } A \in \mathcal{D} \setminus \mathcal{T}; \\ \nu(A)\tau(A), & \text{if } A \in \mathcal{T} \end{cases}$$

$$z_A = \begin{cases} \text{true}, & \text{if } 0 \overset{\circ}{\in} A; \\ \text{false}, & \text{otherwise.} \end{cases}$$

Algorithm 4.1 *Multiplication of Generalised Intervals.*
$A = [a^-, a^+]$, $B = [b^-, b^+]$, $A \times B = R$

$\theta_A = a^+ < a^-$
if θ_A then $A = -A$
$z_A = (a^- < 0 \text{ or } a^+ < 0)$

$\theta_B = b^+ < b^-$
if θ_B then $B = -B$
$z_B = (b^- < 0 \text{ or } b^+ < 0)$

if z_A & z_B then if $(a^+ < 0) \neq (b^+ < 0)$
then $R = [0, 0]$
elseif $(a^+ < 0)$ then $R = [a^-b^-, \max\{a^-b^+, a^+b^-\}]$
else $R = [\min\{a^-b^+, a^+b^+\}, a^+b^+]$
elseif z_A then if $(a^+ < 0)$ then $R = [a^-b^-, a^+b^-]$
else $R = [a^-b^+, a^+b^+]$
elseif z_B then if $(b^+ < 0)$ then $R = [a^-b^-, a^-b^+]$
else $R = [a^+b^-, a^+b^+]$
else $R = [a^-b^-, a^+b^+]$

if $\theta_A = \theta_B$ then Return R
else Return $-R$

To achieve a better lucidity of Algorithm 4.1, only the end-points of the result are specified without explicit referring to a directed rounding. An implementation should provide that left end-point is rounded toward $-\infty$ and right end-point is rounded toward $+\infty$.

For normal intervals, Algorithm 4.1, performed on parallel processors, would have performance comparable to that of Algorithm 3.4. The average slow-down of Algorithm 4.1 would be about 2-3 sign-tests. However, a small implementation and interpretive overhead would be at the expense of an increased functionality of generalised arithmetic.

Other implementation approaches, based on formula (2), seem to be less than optimal since the direction of an extended interval is determined by a floating-point comparison and an interval sign-test involves two floating-point sign-tests.

5 Conclusion

We presented here the theoretical basis of interval multiplication algorithms using at most three floating-point products and discussed several approaches for the implementation of interval multiplication operation. The proposed cost-efficiency related improvements require no special hardware support. We share the opinion of [7] "the final verdict on cost-effectiveness would require comparison of comparable hardware and software implementations" and hope that the approaches, presented in this paper, would contribute toward (or at least would provoke searching) an optimal hardware implementation attracting more interval applications.

References

[1] Alefeld, G.; Herzberger, J.: "Einführung in die Intervallrechnung". Bibliographisches Institut Mannheim (1974). English translation: "Interval Analysis", Academic Press (1981).

[2] American National Standards Institute/Institute of Electrical and Electronics Engineers: "IEEE Standard for Binary Floating-Point Arithmetic"; ANSI/IEEE Std 754–1985, New York (1985).

[3] Chiriaev, D.; Walster, G. W.: "Interval Arithmetic Specification", Draft revised October 1997.
(http://www.mscs.mu.edu/~georgec/Classes/GlobSol/Papers/spec.ps)

[4] Christ, H.: "Realisierung einer Maschinenintervallarithmetik mit beliebigen ALGOL-60 Compilern", Elektron. Rechenanlagen, 10 (1968), 217-222.

[5] Dimitrova, N.; Markov, S.; Popova, E.: "Extended Interval Arithmetics: New Results and Applications", in Atanassova, L.; Herzberger, J. (Eds.):

"Computer Arithmetic and Enclosure Methods", Elsevier Science Publishers B.V. (1992), 225-232.

[6] Heindl, G.: "Improved Algorithm for Computing the Product of Two Machine Intervals", Interner Bericht der integrierten Arbeitsgruppe Mathematische Probleme aus dem Ingenieurbereich Fachbereich Mathematik der bergischen Universität-Gesamthochschule Wuppertal, IAGMPI – 9304, September (1993).

[7] Hough, D.: "Hardware Support for Interval Arithmetic", posts to numeric-interest@validgh.com mailing list, March 1994; October 1995.
(http://www.validgh.com/numeric-interest/numeric-interest.archive)

[8] Kulisch, U., Miranker, W. L.: "Computer Arithmetic in Theory and Practice"; Academic Press, New York (1981).

[9] Popova, E.: "Interval Operations Involving NaNs"; Reliable Computing, No. 2 (1996), 161-166.

[10] Popova, E.: "Generalized Interval Distributive Relations"; Preprint No 2, Institute of Mathematics & Computer Science, Bulgarian Academy of Sciences, February 1997, 1-18.

[11] Popova, E.; Markov, S.: "Towards Credible Implementation of Inner Interval Operations". 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics. Volume 2 Numerical Mathematics; 1997, 371-376.

[12] Schulte, M. J.; Swartzlander, E. E. Jr.: "Software and Hardware Techniques for Accurate, Self-Validating Arithmetic", in: Kearfott, R. B.; Kreinovich, V. (Eds.): "Applications of Interval Computations". Applied Optimization **2**, Kluwer Academic Publishers (1996) 381-404.

[13] Wolff von Gudenberg, J.: "Hardware Support for Interval Arithmetic. Extended Version"; Report No. 125, Lehrstuhl für Informatik II, Universität Würzburg, October 1995, 1-14.
(ftp://sunshine.informatik.uni-wuerzburg.de/pub/publications/tr125.ps)