Quantitative Results

Tomáš Brabec

Abstract

We provide results of a comparison between the speculatively redundant continued logarithms and some other representations based on linear fractional transformations (LFTs). Its objective was to quantify differences in certain parameters among the considered representations. The measured figures indicate that no representation has substantial advantages over the others. We admit, however, that the performed comparison was too simple to generalize its conclusions.

This document accompanies the manuscript [1] that proposes a computable extension of the continued logarithm representation. Our aim is to provide some experimental results that would support a rather theoretical discussion in section “Practical Effects of Redundancy” of the manuscript. Another reason for running the experiments was to find out how the proposed representation compares to some other related representations.

The comparison focused on parameters discussed in the referred section of [1]; Section 3 provides the details. The measurement happened on three simple algorithms (benchmarks, Section 2) that can be computed by the operator of BLFT (bi-linear fractional transformation), which is central to any LFT-based arithmetic. Representations considered in the experiments are described in Section 1.

The results were measured by an automated framework. As the framework is presently in development, the results shall be considered with care and for illustrative purposes only. Implementations of individual representations are stable, but some of them may be better optimized than others. Different stage of optimization could have affected some of the results, especially the performance.

1 Representations

Representations considered for the comparison are listed below. All representations are implemented in Java and they subclass from the same base class that provides most of the general functionality. Individual subclasses then implement only the three step process absorb-decide-emit. This should make the implementations consistent and roughly similar regarding the performance comparison.

- Simple Continued Fractions: An implementation of the basic continued fraction scheme [2], [3]. Elements of the representation are arbitrary integers (through BigInteger Java class) and thus it is the only non-binary representation in this comparison.
- IC Reals: An implementation of a LFT radix scheme used by Heckmann [4]. The binary representation is obtained by using the radix \( r = 2 \).
- RACF: An implementation of the redundant admissible continued fractions proposed by Kornerup and Matula [5].
- RACF (NR): A non-redundant variant of the RACF representation. This is basically a binary encoded implementation of simple continued fractions. This representation borrows from RACF the idea of a unary prefix for delimiting the length of binary encoded partial quotients.
- Continued Logarithms: An implementation of the continued logarithm representation proposed by Gosper [3] and extended to \( \mathbb{R}^* \) as described in [1].
- Speculative Continued Logarithms: An implementation of the representation proposed in [1].

2 Benchmarks

As for the benchmarks, we used the algorithms below. Some of the algorithms were discussed in the main manuscript [1], but the results discussed therein come from [6]. Randomized benchmarks were run on the same set of random data for all representations to make the results consistent.

In general, each benchmark shall be considered as an expression tree, the root of which is to be evaluated to the selected target precision. If there are multiple roots, the precision constraints adhere to all of them.

- Random BLFTs: This benchmark runs a set of BLFTs, i.e. rational functions:
  \[
  f(x, y) = \frac{axy + by + cx + d}{exy + fy + gx + h},
  \]
where \( a \) through \( h \) are integers (called fractional coefficients) and \( x, y \) are arguments. BLFTs were randomly initialized with rational arguments and randomly generated fractional coefficients. The benchmark is parametrized only by the target precision (in bits) to which the BLFT result shall be evaluated.

- **Logistic Map**: This example comes from [7]:
  \[ x_{n+1} = 3.999x_n(1 - x_n), \]
  where \( x_0 = \frac{2}{3} \) and the length \( n \) of the sequence is a parameter of the algorithm. The benchmark is further parametrized by the target precision of the result.

- **Matrix inversion**: This benchmark computes an inverse matrix using the LU decomposition. This is an example of a complex algorithm, control flow of which depends on its intermediate results. The benchmark is parametrized by the matrix dimension, matrix initialization (i.e. matrix elements) and a target precision of elements of the inverted matrix.

### 3 Parameters

The set of observed parameters in this comparison corresponds to those discussed in the main manuscript [1]. These parameters are as follows:

- **Expansion length**: This is the number of elements in the computed representation of the root of an expression tree. In case of matrix inversion where there are multiple roots, this parameter is taken for just one of them. From a practical point of view, the shorter the expansion length the better.

- **Throughput**: This is an average number of absorptions per emission of a BLFT algorithm for the root of an expression tree. This parameter shall quantify the emission rate of each representation. It has some limitation, though, and thus shall be considered for illustration only. A lower value indicates better throughput, and thus a better chance for parallel evaluation of the expression tree.

- **Maximum bit-length**: The bit-length is the number of bits to represent a fractional coefficient of a BLFT operator. The maximum represents the peak bit-length detected for fractional coefficients of the root of a computed expression tree. In practice, the maximum bit-length identifies the size of registers needed to safely compute the desired result. Therefore, the lower the value the better.

- **Total maximum bit-length**: This parameter is the same as above, but this time it is the peak detected over a complete expression tree. One may expect that leaves and inner nodes of an expression tree need to be evaluated to higher precision than the root. Therefore it is expected that the actual peak bit-length will be higher than that of the root.

- **Average time**: This is the basic performance parameter and it is the time in milliseconds it takes to evaluate an expression tree to the required precision of its root(s). The evaluation was run several times and an average value was considered. Lower values indicate better performance.

### 4 Results

The collected data are presented in tables Tab. 1 (matrix inversion), Tab. 2 (BLFTs) and Tab. 3 (logistic map). The actual collecting happened in two phases. In the first phase, the detailed information about evaluation process were obtained by attaching observers to all computed arithmetic expressions. The second phase measured only the performance, when no logging or observing of the computation process happened. The performance measurement was organized so that a benchmark was run several times and an average value was taken.

### 5 Evaluation

This section compares results of certain parameters among individual benchmarks. In particular, we consider parameters of performance (Tab. 4), expansion length (Tab. 6) and total max. bit-length (Tab. 5). The results we compare are those for the target precision of 20 bits, because otherwise the precision varied for each benchmark. This restriction shall be acceptable as the results of individual benchmarks seem consistent with respect to varying input parameters (such as precision and others, depending on the benchmark).

The first considered parameter is the performance. It is the easiest one to comprehend, but the hardest one to measure. There are a number of external forces that can affect the results, e.g. system load, particular order in which benchmarks are executed\(^{1}\), the level of optimization of each implementation, etc. Hence the conclusions based on the measured figures shall be taken with extreme care.

---

1. This is especially true for Java. The garbage collection is delayed and trash resources from a previous run can affect the next run of a benchmark. Additionally, run-time code optimization inside a virtual machine can also affect the overall performance.
<table>
<thead>
<tr>
<th>Matrix Type</th>
<th>Prec.</th>
<th>Representation</th>
<th>Expansion length</th>
<th>Throughput</th>
<th>Maximum bit-length</th>
<th>Total max. bit-length</th>
<th>Average time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random 5</td>
<td>5</td>
<td>Cont. Log.</td>
<td>54</td>
<td>2.30</td>
<td>41</td>
<td>139</td>
<td>368.7</td>
</tr>
<tr>
<td>Random 5</td>
<td>5</td>
<td>IC Reals</td>
<td>33</td>
<td>2.12</td>
<td>37</td>
<td>88</td>
<td>316.1</td>
</tr>
<tr>
<td>Random 5</td>
<td>5</td>
<td>Simple Cont. Frac.</td>
<td>32</td>
<td>2.06</td>
<td>70</td>
<td>219</td>
<td>375.0</td>
</tr>
<tr>
<td>Random 5</td>
<td>5</td>
<td>RACF</td>
<td>52</td>
<td>2.04</td>
<td>32</td>
<td>111</td>
<td>983.8</td>
</tr>
<tr>
<td>Random 5</td>
<td>5</td>
<td>RACF (NR)</td>
<td>58</td>
<td>2.17</td>
<td>37</td>
<td>156</td>
<td>539.5</td>
</tr>
<tr>
<td>Random 5</td>
<td>5</td>
<td>Spec. Cont. Log.</td>
<td>58</td>
<td>2.03</td>
<td>34</td>
<td>144</td>
<td>663.1</td>
</tr>
<tr>
<td>Random 20</td>
<td>5</td>
<td>Cont. Log.</td>
<td>68</td>
<td>2.09</td>
<td>49</td>
<td>160</td>
<td>511.7</td>
</tr>
<tr>
<td>Random 20</td>
<td>5</td>
<td>IC Reals</td>
<td>47</td>
<td>2.04</td>
<td>49</td>
<td>118</td>
<td>457.0</td>
</tr>
<tr>
<td>Random 20</td>
<td>5</td>
<td>Simple Cont. Frac.</td>
<td>59</td>
<td>2.34</td>
<td>135</td>
<td>312</td>
<td>464.8</td>
</tr>
<tr>
<td>Random 20</td>
<td>5</td>
<td>RACF</td>
<td>75</td>
<td>1.97</td>
<td>41</td>
<td>121</td>
<td>1414.2</td>
</tr>
<tr>
<td>Random 20</td>
<td>5</td>
<td>RACF (NR)</td>
<td>76</td>
<td>2.12</td>
<td>43</td>
<td>176</td>
<td>649.4</td>
</tr>
<tr>
<td>Random 20</td>
<td>5</td>
<td>Spec. Cont. Log.</td>
<td>76</td>
<td>2.03</td>
<td>44</td>
<td>159</td>
<td>926.2</td>
</tr>
<tr>
<td>Hilbert 5</td>
<td>5</td>
<td>Cont. Log.</td>
<td>46</td>
<td>n/a</td>
<td>27</td>
<td>55</td>
<td>154.7</td>
</tr>
<tr>
<td>Hilbert 5</td>
<td>5</td>
<td>IC Reals</td>
<td>60</td>
<td>2.07</td>
<td>65</td>
<td>266</td>
<td>548.2</td>
</tr>
<tr>
<td>Hilbert 5</td>
<td>5</td>
<td>Simple Cont. Frac.</td>
<td>1</td>
<td>n/a</td>
<td>17</td>
<td>29</td>
<td>43.4</td>
</tr>
<tr>
<td>Hilbert 5</td>
<td>5</td>
<td>RACF</td>
<td>30</td>
<td>2.50</td>
<td>18</td>
<td>43</td>
<td>282.5</td>
</tr>
<tr>
<td>Hilbert 5</td>
<td>5</td>
<td>RACF (NR)</td>
<td>30</td>
<td>n/a</td>
<td>31</td>
<td>45</td>
<td>141.9</td>
</tr>
<tr>
<td>Hilbert 5</td>
<td>5</td>
<td>Spec. Cont. Log.</td>
<td>49</td>
<td>2.50</td>
<td>27</td>
<td>54</td>
<td>273.4</td>
</tr>
<tr>
<td>Hilbert 20</td>
<td>5</td>
<td>Cont. Log.</td>
<td>46</td>
<td>n/a</td>
<td>27</td>
<td>55</td>
<td>161.1</td>
</tr>
<tr>
<td>Hilbert 20</td>
<td>5</td>
<td>IC Reals</td>
<td>71</td>
<td>2.08</td>
<td>77</td>
<td>266</td>
<td>682.2</td>
</tr>
<tr>
<td>Hilbert 20</td>
<td>5</td>
<td>Simple Cont. Frac.</td>
<td>1</td>
<td>n/a</td>
<td>17</td>
<td>29</td>
<td>52.4</td>
</tr>
<tr>
<td>Hilbert 20</td>
<td>5</td>
<td>RACF</td>
<td>30</td>
<td>2.50</td>
<td>18</td>
<td>43</td>
<td>275.3</td>
</tr>
<tr>
<td>Hilbert 20</td>
<td>5</td>
<td>RACF (NR)</td>
<td>30</td>
<td>n/a</td>
<td>31</td>
<td>45</td>
<td>138.2</td>
</tr>
<tr>
<td>Hilbert 20</td>
<td>5</td>
<td>Spec. Cont. Log.</td>
<td>49</td>
<td>2.50</td>
<td>27</td>
<td>54</td>
<td>283.4</td>
</tr>
</tbody>
</table>

**TABLE 1**
Summary of observed parameters measured for the 5x5 matrix inversion benchmark.

<table>
<thead>
<tr>
<th>Prec.</th>
<th>Representation</th>
<th>Expansion length</th>
<th>Throughput</th>
<th>Maximum bit-length</th>
<th>Total max. bit-length</th>
<th>Average time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Cont. Log.</td>
<td>13</td>
<td>2.31</td>
<td>47</td>
<td>64</td>
<td>0.367</td>
</tr>
<tr>
<td>10</td>
<td>IC Reals</td>
<td>13</td>
<td>2.23</td>
<td>58</td>
<td>94</td>
<td>0.979</td>
</tr>
<tr>
<td>10</td>
<td>Simple Cont. Frac.</td>
<td>4</td>
<td>3.75</td>
<td>58</td>
<td>74</td>
<td>0.296</td>
</tr>
<tr>
<td>10</td>
<td>RACF</td>
<td>12</td>
<td>3.25</td>
<td>46</td>
<td>57</td>
<td>1.304</td>
</tr>
<tr>
<td>10</td>
<td>RACF (NR)</td>
<td>13</td>
<td>2.38</td>
<td>43</td>
<td>57</td>
<td>0.489</td>
</tr>
<tr>
<td>10</td>
<td>Spec. Cont. Log.</td>
<td>14</td>
<td>2.14</td>
<td>47</td>
<td>58</td>
<td>0.605</td>
</tr>
<tr>
<td>20</td>
<td>Cont. Log.</td>
<td>25</td>
<td>2.16</td>
<td>57</td>
<td>67</td>
<td>0.537</td>
</tr>
<tr>
<td>20</td>
<td>IC Reals</td>
<td>23</td>
<td>2.17</td>
<td>79</td>
<td>113</td>
<td>1.607</td>
</tr>
<tr>
<td>20</td>
<td>Simple Cont. Frac.</td>
<td>6</td>
<td>2.83</td>
<td>58</td>
<td>81</td>
<td>0.413</td>
</tr>
<tr>
<td>20</td>
<td>RACF</td>
<td>22</td>
<td>2.32</td>
<td>49</td>
<td>59</td>
<td>1.982</td>
</tr>
<tr>
<td>20</td>
<td>RACF (NR)</td>
<td>25</td>
<td>1.92</td>
<td>49</td>
<td>61</td>
<td>0.736</td>
</tr>
<tr>
<td>20</td>
<td>Spec. Cont. Log.</td>
<td>28</td>
<td>1.93</td>
<td>57</td>
<td>66</td>
<td>1.113</td>
</tr>
<tr>
<td>50</td>
<td>Cont. Log.</td>
<td>56</td>
<td>2.00</td>
<td>75</td>
<td>97</td>
<td>1.109</td>
</tr>
<tr>
<td>50</td>
<td>IC Reals</td>
<td>53</td>
<td>2.12</td>
<td>137</td>
<td>152</td>
<td>3.882</td>
</tr>
<tr>
<td>50</td>
<td>Simple Cont. Frac.</td>
<td>14</td>
<td>2.64</td>
<td>73</td>
<td>97</td>
<td>0.692</td>
</tr>
<tr>
<td>50</td>
<td>RACF</td>
<td>56</td>
<td>1.98</td>
<td>65</td>
<td>76</td>
<td>4.075</td>
</tr>
<tr>
<td>50</td>
<td>RACF (NR)</td>
<td>55</td>
<td>1.98</td>
<td>65</td>
<td>76</td>
<td>1.459</td>
</tr>
<tr>
<td>50</td>
<td>Spec. Cont. Log.</td>
<td>66</td>
<td>1.67</td>
<td>80</td>
<td>93</td>
<td>2.151</td>
</tr>
</tbody>
</table>

**TABLE 2**
Summary of observed parameters measured for the random BLFT benchmark.
Anyway, the results illustrate that non-redundant representations exhibit higher performance than redundant ones. As discussed in the manuscript, the redundancy in continued logarithms shall worsen the performance roughly twice. The actual implementation of the speculative continued logarithms differs slightly from this expectation, making the worst case performance degrade by at most four times\(^2\). The same holds also for RACF compared to RACF NR.

Interesting results were obtained for simple continued fractions. This is the only non-binary representation in this comparison and its results show that optimizing computer algorithms for add-shift operations may not always pay off. The implementation of simple continued fractions uses all the BigInteger operations including multiplication and division, and still the performance comes better than with most binary representations. One reason can be seen in that the binary representations have more complex decision algorithms (in the sense of different if-else branches) than simple continued fractions; the length of expansion may have its effect too (see below).

Another interesting result, which requires further investigation, is a seeming dependence on the type of input values. At least the comparison between Hilbert and random matrix indicates so. Hilbert matrix could be considered a sort of integer matrix, and thus continued fractions favor it, especially the simple continued fractions. IC reals exhibited in this case much worse performance than continued fraction representations. On the other hand, IC reals showed better performance for random matrices.

The peak bit-length is a parameter that indicates how eligible a particular representation is for hardware implementation. Lower bit-length would require shorter registers and thus saves occupied chip area. According to our discussion in the manuscript, a redundant representation shall have lower demands on the bit-length of fractional coefficients. One would also expect that a binary representation shall be more adequate than a non-binary one, especially because of its finer granularity. This is not always true as witnessed by the results of simple continued

\(^2\) The maximum input domain of a BLFT operator is \(\pm B \times \pm B\), which has 9 distinct “corners” where we need to evaluate functional values of a computed BLFT. Instead, the actual implementation evaluates the functional values quadrant by quadrant (i.e. \(+B \times B\), \(-B \times -B\), etc.), resulting into 16 “corner” values. The decision step of the BLFT algorithm (see Alg. 1 in [1]) of the speculative representation thus evaluates 4 times more values than the decision step of the non-redundant continued logarithms.
fractions.

The last considered parameter is the expansion length. The number of elements that has to be processed can affect the overall computation time quite substantially. This likely makes the difference in performance between simple continued fractions and their binary alternatives. The simple continued fractions have typically short expansions, which means that the overall evaluation algorithm needs to be executed less often.

The manuscript reported that the speculative representation exhibited longer expansions for results computed exactly. It was guessed, however, that the differences shall diminish when evaluating to an approximate value with some fixed precision. The results presented here do not support that claim and the speculative continued logarithms still produce longer expansions.

\begin{table}
\centering
\begin{tabular}{|l|c|c|c|c|c|c|c|}
\hline
BLFT & Bit length & Logistic Map (length = 20) & Bit length & Random Matrix & Bit length & Hilbert Matrix & Bit length \\
\hline
RACF & 59 & RACF & 65 & IC Reals & 118 & Simple Cont. Frac. & 29 \\
RACF (NR) & 61 & Spec. Cont. Log. & 69 & RACF & 121 & RACF & 43 \\
Simple Cont. Frac. & 81 & RACF (NR) & 88 & RACF (NR) & 176 & Cont. Log. & 55 \\
IC Reals & 113 & IC Reals & 177 & Simple Cont. Frac. & 312 & IC Reals & 266 \\
\hline
\end{tabular}
\caption{Comparison of a total maximum bit-length of fractional coefficients over all benchmarks.}
\end{table}

\begin{table}
\centering
\begin{tabular}{|l|c|c|c|c|c|c|}
\hline
BLFT & Expan. length & Logistic Map (length = 20) & Expan. length & Random Matrix & Expan. length & Hilbert Matrix & Expan. length \\
\hline
Simple Cont. Frac. & 6 & Simple Cont. Frac. & 6 & IC Reals & 47 & Simple Cont. Frac. & 1 \\
RACF & 22 & Cont. Log. & 22 & Simple Cont. Frac. & 59 & RACF & 30 \\
IC Reals & 23 & RACF (NR) & 22 & Cont. Log. & 68 & RACF (NR) & 30 \\
RACF (NR) & 25 & IC Reals & 22 & RACF & 75 & Cont. Log. & 46 \\
\hline
\end{tabular}
\caption{Comparison of an expansion length over all benchmarks.}
\end{table}

6 CONCLUSIONS

Finally, let us summarize the hypotheses proposed in [1] (Section “Practical Effects of Redundancy”) and reflect their validity on the quantitative results here.

- **Hypothesis**: Speculative redundancy offers higher responsiveness (throughput) than non-redundant continued logarithms.
  \textit{Validity}: Seems to hold, but the improvement is mostly negligible.

- **Hypothesis**: Speculative redundancy requires less bits for fractional coefficients than non-redundant continued logarithms.
  \textit{Validity}: Seems to hold, but the significance of the claim varies for different benchmarks.

- **Hypothesis**: Speculative redundancy exhibits longer expansions than non-redundant continued logarithms. This may change when not evaluating results to their exact value.
  \textit{Validity}: Does not hold. The speculatively redundant representation seems to have always longer expansions.

- **Hypothesis**: A fully redundant continued logarithm representation shall have roughly twice\(^3\) worse performance than the non-redundant representation. Speculative redundancy shall be somewhere in between.
  \textit{Validity}: This hypothesis cannot be verified as an implementation of a fully redundant continued logarithms is presently missing. The performance degradation measured for speculative representation varied somewhere around two.

\(^3\) As explained here in Section 5, the expected worst case performance degradation is really 4. This regards only the decision step, though, and thus the impact on the overall algorithm performance shall be lower.
There are some other hypotheses that we originally expected, either based on the preliminary results in [6] or on known properties of individual representations:

- **Hypothesis**: RACF representation has shorter expansions and requires less bits for fractional coefficients than speculative continued logarithms. It would thus exhibit better performance.
  **Validity**: The claim about expansion length and size of fractional coefficients seems to hold, but the witnessed performance is worse. The difference in expected performance is yet to be investigated.

- **Hypothesis**: Continued fractions shall have better approximation properties than radix-like representations. One would thus expect IC reals to have longer expansions. In the sequel, IC reals would also be less efficient in other observed parameters.
  **Validity**: Does not hold. In fact, IC reals seem well competitive (except the case of Hilbert matrix) to other binary redundant representations.

- **Hypothesis**: Simple continued fractions are less effective regarding fractional coefficients size than their binary alternatives.
  **Validity**: Does not seem to hold. This is illustrated by the difference in results of simple continued fractions and RACF (NR). This is interesting even more as the latter representation shall be but a binary encoded form of the former.

**REFERENCES**


