Improved Data and Instruction Locality in Long Division

Isaac Grosof
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
`igrosof@cmu.edu`

Isaac Grosof
Computing Hardware

March 18, 2022

Abstract

Long division, as with many algorithms with "long" on the name, suffers from poor practical performance across a wide variety of computing hardware (schoolchildren, typically). We propose a novel implementation of a long-division-style algorithm with greatly improved data and instruction locality, leading to superior performance over the traditional algorithm, particularly in the asymptotic regime.

1 Introduction

Long division is taught to millions of children across the world [7]. The algorithm dates back 1149, first given in "The Shining Book on Calculation" by Ibn Yahya al-Maghribi al-Samaw'al [9, 1]. Its modern incarnation was given in Henry Brigs around 1600 [2].

Unfortunately, the traditional long-division implementation has needlessly poor instruction and data locality, leading to poor performance [4]. It shares this poor performance with many algorithms that feature "long" in the name, such as the dreaded "long multiplication", which prior work has shown should be replaced by lattice multiplication [5].

Motivating the need for an improved long division algorithm are programs written by Matt Parker, who routinely employs long-division within complex computations. These computations often exhibit poor results despite enthusiastic computational hardware [8], as compared to prior work using (presumably) superior algorithms [10].

2 Failings of traditional long division

To improve long division, we must first understand the sources of its poor performance. In principal, performance need not be poor, as the algorithm requires only a constant number of operations per output digit. Nonetheless, in Fig. 1, we see the remains of the typical long-division misadventure.

The poor data locality is evident in the quadratic use of paper and/or screen space. In the asymptotic limit (which is all that matters in algorithm design), the runtime will be dominated by moving back and forth between disconnected parts of the workspace, exhibiting horrendous data locality. This poor data locality is a symptom of premature optimization: The result of each subtraction operation is placed in its traditional location, under the multiplication result, forcing a down-and-left data movement over the course of execution. Our algorithm removes this premature optimization, resulting in a major improvement.

Poor instruction locality is more subtle, but no less egregious. The hardware is expected to constantly cycle between four operations:

- Multiplication
- Subtraction

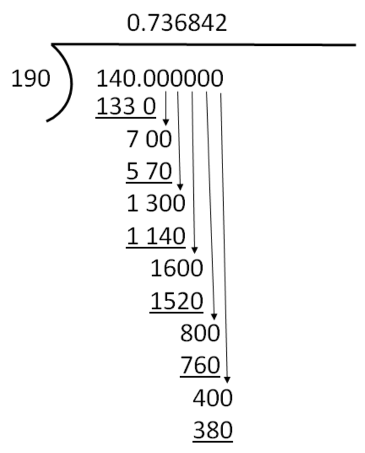

Figure 1: The traditional long-division algorithm [3]

.

- Comparison (checking that the multiple of the divisor produced is the largest that is smaller than the current quotient).
- Recording the output

By constantly wiping the instruction cache, the hardware is forced to continually re-access the algorithm for the specific desired operation, wasting precious cycles.

3 Aside: Short division

A rarely used alternative to long division is *short division* [6], which improves upon the poor data locality of traditional long division, but at the cost of extreme register pressure, overtaxing typical hardware's short-term memory to the point where errors become common, and double checking is required.

In addition, short division requires higher data density, requiring writing between the digits of previously written numbers. Such density requirements run into hardware limitations and readback fidelity issues, again worsening performance. Worse yet, density requirements scale with the size of the dividend, becoming wholy unreadable in the asymptotic limit.

Short division also does nothing to alleviate the poor instruction locality of long division.

While short division presents an interesting alternative to long division, it too suffers from similar failings, despite the ambitious name.

4 Our implementation of long division

We proceed via the following steps:

- Create a lookup table caching the multiplicand multiplied by each digit. Repeated addition can be used in place of multiplication.
- Perform the following pair of steps repeatedly:
  - Perform the "compare and subtract" operations, but write each difference to the right of the previous difference, rather than below, as is traditional.
  - Append the next digit of the quotient to the difference.
- When the termination condition is reached, record the output.

A sample execution performed on the second author is shown in Fig. 2, showing the computation of $\frac{1}{29}$ until repetition.

The improvement in data locality is enormous - the distance between the inputs and outputs of a given computational step is typically a single cell (using king's adjacency), with the only common exception being the multiplication table.

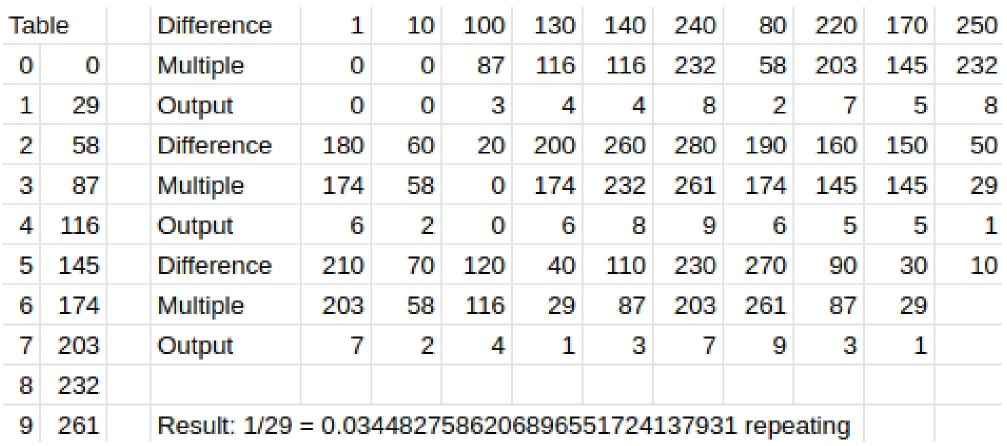

Table			Difference	1	10	100	130	140	240	80	220	170	250
0	0		Multiple	0	0	87	116	116	232	58	203	145	232
1	29		Output	0	0	3	4	4	8	2	7	5	8
2	58		Difference	180	60	20	200	260	280	190	160	150	50
3	87		Multiple	174	58	0	174	232	261	174	145	145	29
4	116		Output	6	2	0	6	8	9	6	5	5	1
5	145		Difference	210	70	120	40	110	230	270	90	30	10
6	174		Multiple	203	58	116	29	87	203	261	87	29	
7	203		Output	7	2	4	1	3	7	9	3	1	
8	232												
9	261		Result: 1/29 = 0.0344827586206896551724137931 repeating										

Figure 2: Our long division implementation, used to exactly compute $\frac{1}{29}$

Likewise, we demonstrate a major improvement in instruction locality. Within the hot loop of the program, the only operations performed are "compare and subtract", removing both the complicated multiplication operation, as well as the output operation, which formerly required massive data movement.

Notably, these improvements are achieved without any new overheads. In particular, neither register pressure nor data density requirements are increased beyond that exhibited by the traditional long division implementation.

5 Experimental results

The test hardware showed much performance and more enjoyment using our revised algorithm. For further verification, we intend to port our algorithm to a wide variety of computational hardware, such as a bright five-year old.

6 Conclusion

We developed a novel implementation of the long division algorithm, achieving far superior performance through improved instruction and data locality. We recommend that this implementation replace the traditional long division algorithm, overthrowing centuries of educational tradition in one stroke. In future work, we plan to overturn more of the arithmetic curriculum, there's probably lots of other outdated algorithms in there.

References

[1] Ibn Yahya al-Maghribi al-Samaw'al. *The Shining Book on Calculation.* 1149.

[2] Henry Briggs. *Oxford Reference.*

[3] Chad Flinn and Mark Overgaard. *Math for Trades: Volume 1.* BCcampus, 2020.

[4] Isaac Grosof. Personal experience, 2003.

[5] Isaac Grosof and Isaac Grosof. On the time complexity of the verification of the factorization of $2^{67} - 1$. *SIGBOVIK*, April 2019.

[6] Lenore John. The effect of using the long-division form in teaching division by one-digit numbers. *The Elementary School Journal*, 30(9):675–692, 1930.

[7] David Klein and R James Milgram. The role of long division in the K–12 curriculum, 2000.

[8] Matt Parker. Can we calculate 100 digits of π by hand? The William Shanks method.

[9] Liz Rogers. Islamic mathematics. August 2008.

[10] William Shanks. On the extension of the numerical value of π. *Proceedings of the Royal Society of London*, 21(139-147):318–319, 1873.