Eager Recirculating Memory to Alleviate the Von Neumann Bottleneck

Jonathan Edwards
York Centre for Complex Systems Analysis,
University of York,
York, UK

Simon O'Keefe
York Centre for Complex Systems Analysis,
University of York,
York, UK

***Abstract*— This paper presents an examination of channel based time delays and their application as units which perform storage and computation. We describe the implementation of compound arithmetic operations, and show that by recirculating the impulses along a channel, both memory and computation can be achieved on the same general channel unit. In addition, this approach has the further advantage of performing arithmetic simplification eagerly, so that the resultant use of memory is optimised by the intermediate processing during memory circulation phases.**

I. Introduction

Central Processor Unit (CPU) design is fixated on binary data storage. This is not surprising since the development of powerful computation devices has become an integral part of the technological advancement of our modern society. Edwards *et al* [1] take a step back from this approach, exploring an alternative method of performing computation without the storage and manipulation of binary data. The method proposed uses time delays across a channel as a representation of numbers and as a computation medium, and in [1] this system is shown to have the capacity to perform all forms of arithmetic computation.

This paper examines this approach in more detail, presenting a *recirculation* method that allows us to store arithmetic operations on the channel. This allows processing to occur *on the same medium as is used for storage*, effectively removing the "von Neumann Bottleneck" (explained in more detail in Section II) as storage of operands and operator and application of that operator use the same mechanism. Furthermore, since the storage medium is actively recirculating, operations may be processed and simplified in an *eager* fashion (that is, the operator may be applied at any time before the result is required), so that the recirculated result is optimised for when it is externally accessed.

The paper is structured as follows: section II reviews the von Neumann Bottleneck to assess it's impact on decreasing CPU operation throughput. To provide the necessary background in time delay computation, section III summarizes the work presented in [1]. This section then expands on this previous work, to demonstrate that non-trivial computation can be performed, and discusses the arrangement of compound arithmetic operations such that they can can be processed by a single stack-less pass across a channel. The fourth section presents a discussion of how these signals on a channel might be recirculated using the addition of a *compute flag* to communicate with other processing units and clocks. The paper concludes with a discussion of the continued development of the algorithmic capabilities of the single time based processing units.

II. The von Neumann Bottleneck

The **von Neumann Bottleneck** is a term coined by John Backus in his 1978 Turing Award Lecture [2]. It describes the imbalance between the speed of computation and the speed of memory access in CPUs designed using the von Neumann architecture. The imbalance arises because the speed with which data is acquired from Random Access Memory (RAM) is significantly lower than the speed at which a CPU can perform computation. This may mean that *wait* cycles are required while data is acquired, ultimately increasing overall computation time. Non-parallel solutions are mainly defined in terms of pipelining, enhanced caching and branch prediction methods [3]. Alternative architectures such as the **Harvard Architecture** [4] also manipulate the nature of memory by providing programme and data stores accessed via separate data buses, that may be of different widths. To a large extent the current directions in chip development have masked this problem, by using ever larger caches and ever smaller elements on a chip, in line with *Moore's Law* [5], to increase computation throughput. Comparable areas of research occur in highly distributed parallel systems, such as brain simulation [6] and neuromorphic hardware [7]. Neuromorphic hardware, and in particular examples implemented in analog, such as IBM's Truenorth processor [8], and on digital channels photonically [9] have similar goals to our work, however, the computational units are often designed to specifically mimic neuronal spike-timing operations rather than our hybrid, initially simple, arithmetic instruction set.

III. Time Delay Processing

We define *channel computation* as the arrangement of data into a signal which is communicated between an encoder and a decoder, via some channel. A channel may be any communication medium between the encoder and decoder, in the sense of Shannon [10]. The data can represent both operators and their operands, and communication via the channel facilitates an efficient decoding, resulting in the evaluation of the computation encoded in the data.

A discrete stream of these data (which can be loosely thought of as a program) can exist on one channel, and the system $encoder, channel, decoder$ (see Figure 1) can

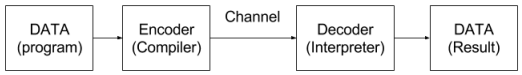

Fig. 1. The channel computation arrangement

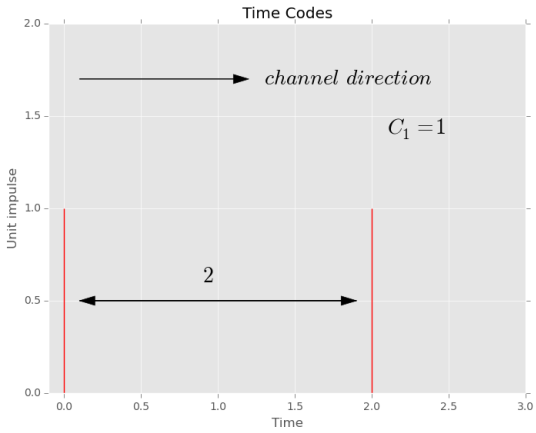

Fig. 2. A Time Delay Unit representing the value 2 as a delay between two impulses.

be thought of as a unit of processing, with the encoder and decoder described by suitable finite state machines (implicitly with some requisite digital *state*).

Edwards *et al* [1] propose an encoding approach in this domain which utilises the temporal aspect of the communication to represent the data as a time delay between individual impulses (see Figure 2). This encoding bears some similarity to various schemes, including Pulse Width Modulation (PWM) [11] and Action Potentials [12].

It is shown then that addition can be performed by the concatenation of two values (see Figure 3) and this is easily extended to subtraction and comparison (See [1], section 3.2). The simplicity of this approach suggests that the encoding and decoding finite state machines will be quite small, which in turn suggests an advantage in real-world instantiation (which we discuss further in section V).

Furthermore, by the inclusion of two clocks measuring the relative speed of the impulses traversing the channel we can extend the set of arithmetic operations available to include multiplication and division. The term "clock" in this context is a device measuring units of time, which subdivides

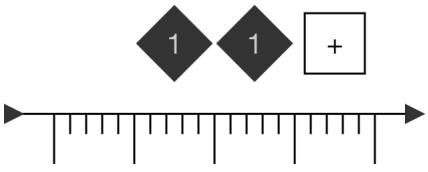

Fig. 3. Simple Addition can be performed by concatenation. This more abstract diagram can be read right to left, and shows the addition operator applied to processing two operands of magnitude 1. This would result in a single output on the channel of magnitude 2.

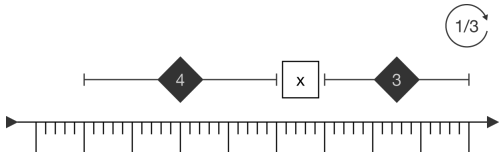

Fig. 4. Multiplication can be implemented using changes to the clock speed. The clock speed is shown at the top right of the diagram, so 1 cycle on this channel represents 3 to an external timer.

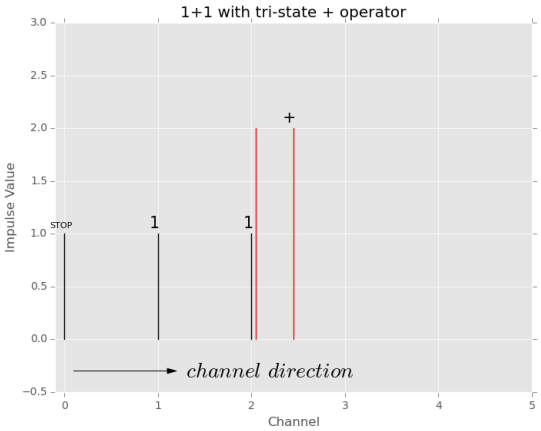

Fig. 5. Instructions can be coded as a separate state on the channel.

the channel discretely into absolutely equivalent time-steps within the encoder and decoder. Also implicit are the notions that the clocks start synchronised and equivalent, and that a mechanism exists to scale their time steps, effectively changing the meaning of a unit time-step at each end the channel. This is illustrated in Figure 4.

A. Representing Operands and Operators

An important design consideration for this system is to develop a form of representation for *operators*. The most straightforward method is to extend the representation so that it is a **tri-state channel** (for example with states labelled 0, 1, and 2). The magnitude of an operand is represented by the time delay between impulses of value 1, and an operator is represented by the time delay between impulses of value 2. This is illustrated in Figure 5. A suitable (simple) FSM is necessary to map between the numerical magnitude representing the computation, and the actual operation.

B. Performing Compound Operations

1) Addition and Subtraction: A requirement for this computation system is that it should utilise the least theoretical "hardware" to perform calculations. There is some subtlety to the way compound operators are stored on the channel as there is no stack to store values from intermediate computations. Figure 6 illustrates this for the concatenation of *addition* operators.

In evaluating an expression such as $2+3+3$, the processing unit requires *intermediate storage* to compute the compound expression. To resolve this difficulty, a simple modification of the position of the operator is required. For the system

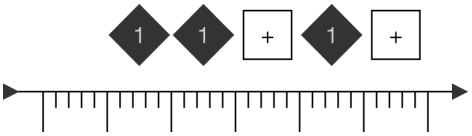

Fig. 6. Addition requires some form of internal memory when performed using an intermediate infix notation.

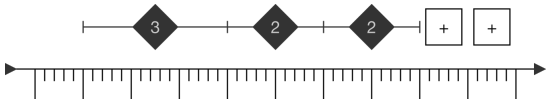

Fig. 7. Addition is a natural process, with a simple algorithm, when represented in prefix notation.

to perform addition no intermediate storage is required, so long as the addition/subtraction operators are arranged in a **prefix** manner. The processing algorithm for addition and subtraction then counts the number of operators and concatenates accordingly. A simple working example (in Figure 7) demonstrates compound addition of $2 + 2 + 3$.

After observing the addition operators, the algorithm starts timing at the first data impulse and then simply ignores the corresponding number of data impulses (thus implementing addition operators) past this point. It stops timing when the subsequent data impulse arrives. So for example, a prefix of $++$ would result in skipping two impulses.

The process is similar for subtraction, with the same labeling method as proposed in the original paper ([1] section 3.2). Figure 8 explains the compound arithmetic expression $4 - 2 + 1$ graphically. The $-$ operator causes the FSM to start timing at the second data impulse (corresponding to the end of the operand 2), so the interval measured to the third data impulse is $4 - 2$. The $+$ operand then causes the FSM to ignore the third impulse, so we have $4 - 2 + 1$.

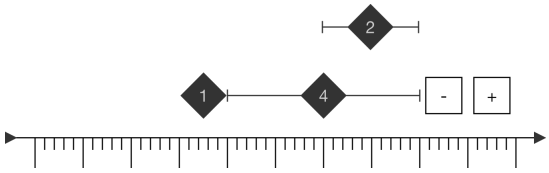

Fig. 8. Subtraction is also easily performed when specified as a prefix operation. We represent the subtraction operands as parallel magnitudes for notational simplicity - the resultant calculation requires a single channel only.

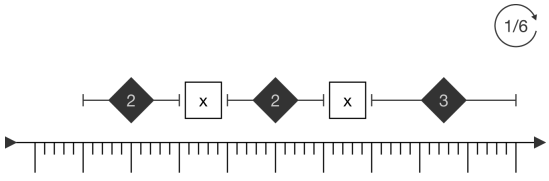

Fig. 9. Multiplication is naturally performed using infix. The first value is moved to the speed register to affect the multiplication. This illustrates evaluation of the expression $3 \times 2 \times 2$.

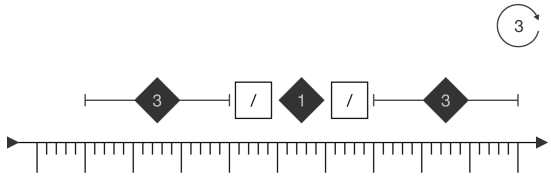

Fig. 10. Division is similar in process to multiplication. This illustrates evaluation of the expression $3/1/3$.

2) Multiplication and Division: Multiplication and Division are also possible with only clock speed modification. The most suitable position of the operator is **infix** as again this removes the necessity of the intermediate storage. The one caveat is that values must pass to the clock register before exiting the channel. A simple worked example evaluating the expression $3 \times 2 \times 2$ is shown in Figure 9. This is performed without the need for intermediate storage of operands and operators. Likewise division is similarly scaled. A simple worked example evaluating the expression $3/1/3$ is shown in Figure 10.

As a final summary, Figure11 collects all the ideas above and demonstrates an arithmetic operation with arbitrary complexity, in this case $3 + 4 \times (6 + 5/(3 - 2))$.

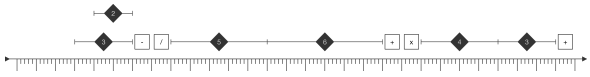

Fig. 11. Complex operations can be performed on a single channel.

IV. Eager Recirculating Memory

The section above expands the description of the computational qualities of the arithmetic processes described in the original paper. This computational process can be further modified to become a memory system. This is achieved by the provision of structure allowing *recirculation* of the impulses. The simplest structure is to connect two channels in a circular fashion to form a closed loop, with impulses recirculated around the loop indefinitely. The computation from this loop can be exposed by the further addition of a **computation switch** which transfers the data *out of the loop*. Figure 12 represents this recirculation graphically.

When the switch is toggled and the circuit becomes open, computation occurs and the data emerges from the channel. Significantly, computation and storage are held *together on the same channel*.

One interesting aspect of the model is that, during recirculation, computation can be silently performed, progressively

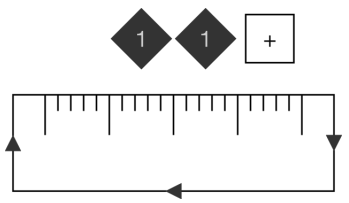

Fig. 12. Memory is implemented using recirculation of the impulses.

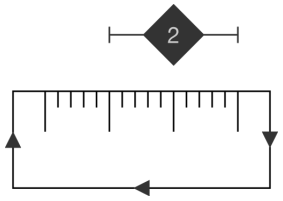

Fig. 13. Instructions can be simplified whilst circulating in memory.

simplifying the data stored in memory. This performs optimisation of the stored expression for free, and simplifies the memory constraints by removing any processable impulses prior to being made externally available through the setting of the computation switch. The simplest example application of this *eager memory* is the addition of $1 + 1$. This is the shown in Figure13, and is trivially simplified to 2 by one circulation of the impulses around the channel loop.

Addition and subtraction operations only manipulate the actual signal on the channel, hence are amenable to simplification. In contrast, multiplication is more problematic since it affects the speed of the channel, and produces multiplicatively larger operands which, as they are temporally encoded, increase the amount of time necessary for computation. Instead, it may (and we are actively investigating) be more efficient to leave the operands in their unoptimised pre-calculation format as multiplicative tuples. An example of this is shown in Figure 14.

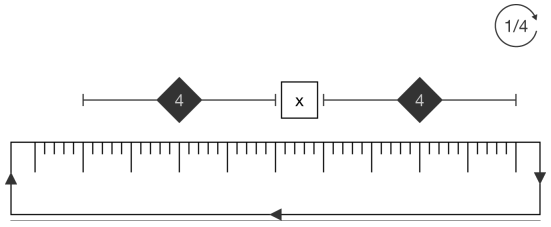

Fig. 14. The intermediate relative clock speeds used in multiplication may prohibit the optimisation of an expression held on the processing unit.

So, the output value is only evaluated when the computation is directly decoded into its final (possibly digital) form. The enticing by-product of this is that large numbers may be encoded as multiplications, so the system has an implicit computational compression scheme, although care must be taken with prime numbers (which we are currently investigating as a part of our further research).

The idea of *laziness* is not a new concept in the field of language design and computation. Several modern languages, particularly functional ones, operate an implicit laziness; holding on to computation without extending them out into memory. Even popular multi-paradigm languages like Python have a `yield` operator which stalls computation until a `do` step.

V. Conclusion and Future Developments

Computation using time as an analogue for magnitude of value offers an alternative theoretical strategy to traditional digital computation. In this paper we have expanded the model to incorporate more compound operations and discussed how they might be held in memory. Furthermore, we have demonstrated that individual time-based units can perform *both* memory and computation and with a minimal modification to the originally framework presented in [1]. Using the scheme we discuss above, we have no differentiation between memory unit and computation unit, and this lack of bottleneck has the added advantage of allowing intermediate expression optimisation during storage.

The next step is to build a more general instruction set that incorporates "Minimum Instruction Set Computing" (MISC)-type instructions, with the aim of building a fully Turing complete, universal processing unit. This is likely to involve the linking of several individual units of the type discussed here into more powerful arrays such that sequence, selection and iteration can occur as a part of inter-channel unit communication. It is also hoped that future research will assess the nature of optimising memory and demonstrate processes that may benefit from its application. It may be possible to perform a system warm-up whilst loading the data into memory, so that effectively much of an algorithmic process has already been worked through prior to program inception. This is effectively a compilation phase for a more extensive time-based processing system. One can speculate on the substrate for a practical hardware system, as time is a channel independent property. At present channels are most readily implemented photonically [13], however microprocessor level integration of our model, to enable computation at "the speed of light" would be limited by the resolution and accuracy of a suitably fast clock.

VI. Acknowledgments

The authors would like to thank Mark Hill, for help in preparation of the diagrams.

References

[1] J. Edwards, S. O'Keefe, and W. D. Henderson, "Unconventional arithmetic: A system for computation using action potentials," in *Proc. of Unconventional Computation and Natural Computation*, 2014, pp. 155–163.

[2] J. Backus, "Can programming be liberated from the von neumann style?: A functional style and its algebra of programs," *Commun. ACM*, vol. 21, no. 8, pp. 613–641, Aug. 1978. [Online]. Available: http://doi.acm.org/10.1145/359576.359579

[3] J. P. Shen and M. H. Lipasti, *Modern processor design : fundamentals of superscalar processors*. Boston: McGraw-Hill Higher Education, 2005, index. [Online]. Available: http://opac.inria.fr/record=b1129703

[4] B. Cohen, *Howard Aiken, Portrait of a computer pioneer*. The MIT Press, 2000.

[5] G. E. Moore, "Readings in computer architecture," M. D. Hill, N. P. Jouppi, and G. S. Sohi, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, ch. Cramming More Components Onto Integrated Circuits, pp. 56–59. [Online]. Available: http://dl.acm.org/citation.cfm?id=333067.333074

[6] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. Ieee, 2008, pp. 2849–2856.

[7] C.-S. Poon and K. Zhou, "Neuromorphic silicon neurons and large-scale neural networks: challenges and opportunities," *Frontiers in neuroscience*, vol. 5, p. 108, 2011.

[8] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[9] A. N. Tait, M. A. Nahmias, Y. Tian, B. J. Shastri, and P. R. Prucnal, "Photonic neuromorphic signal processing and computing," in *Nanophotonic Information Physics*. Springer, 2014, pp. 183–222.

[10] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[11] W. Maass and C. M. Bishop, Eds., *Pulsed Neural Networks*. Cambridge, MA, USA: MIT Press, 1999.

[12] J. J. Hopfield, "Pattern recognition computation using action potential timing for stimulus representation," *Nature*, vol. 376, no. 6535, pp. 33–36, 1995.

[13] A. G. Bakaoukas, "An all-optical soliton fft computational arrangement in the 3nlse-domain," in *Proceedings of the 15th International Conference on Unconventional Computation and Natural Computation - Volume 9726*, ser. UCNC 2016. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 11–24. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-41312-9_2