Rough Draft

Colbyn Wadman

March 22, 2021

I've decided to change my topic of interest. For the reasons given below (will probably be seen in the resulting paper).

In his video essay, A Future of Medicine, Dr. Bernard postulates on a world that views our current system of evidence based medicine, as we see European plague doctors. Because in our modern system of evidence based medicine, we extrapolate population data to a given person, for the purpose of predicting outcome. Which you may ask, what is the problem with such?

To explain, from a practical standpoint, consider this. What if, instead of running a trial on thousands of disperse persons. What if you were simply cloned, hundreds of thousands of times, and therein, we performed hundreds of thousands of trials, forming a stochastic model that likewise predicts outcome for some given treatment?

As Dr. Bernard points out, if a trial was done on a bunch of men, there have been documented cases where women taking the very same medication experienced different reactions. Nowadays, trials are required to include notable age, gender, and race/ethnicity demographics. But as Kravitz pointed out, "[this] may do nothing but ensure that the estimates for any one subgroup are unreliable due to small numbers". In the very same paper, Kravitz coined this phenomena, the "Heterogeneity of Treatment Effects".

Furthermore, Dr. Bernard likewise points out that, if this were possible, we could likewise extend this to drug development. Perhaps as I'd imagine, in a manner that may be inconceivable to us, as I will explain.

But you may ask, how exactly do we run hundreds of thousands of trials of my identical clones? We run

computer simulations, as Dr. Bernard proposed. Although what Dr. Bernard proposed, was essentially a thought experiment, explicitly regarding this as something that probably won't be viable in our lifetime. Conversely, the following paper will say otherwise.

This all began from a TedX talk by a man called Rahul Sarpeshkar (Professor of Engineering, Thomas E. Kurtz Professor, Professor of Microbiology & Immunology, Professor of Physics, Professor of Molecular & Systems Biology). In his talk, Sarpeshkar discussed the prospects of analog computing for solving differential equations in a manner that far outpaces the capabilities of digital computers, and furthermore, he claimed that we could viably simulate an entire person if we built an analog computer that spanned the size of the given auditorium.

Because, as he pointed out, "Digital computation is a subset of analog computation that only operates at its saturated high or low extremes. Therefore, by allowing operation over the whole range of signal levels and by allowing exploitation of all the basis functions of biochemistry and biophysics, not just logic, horizons for computation are expanded. Digital computation will continue to be important for decision-making, signal restoration, communication and sequential operation."

This is because, for instance,

> "[the] fundamental reason for the analog–digital crossover does not lie in issues having to do with parameters and numbers. It really lies in information theory: information is coded across many 1-bit-precise interacting computational channels in the digital approach but on 1 multi-bit computational channel in the analog approach [14]. At low informational precision, logic basis functions simply cannot compete with the richer basis functions of analog computation that can process all the bits at once in parallel and just automatically solve the task, e.g. by using Kirchoff's current law for addition or chemical binding for multiplication."

Furthermore, to further expand upon the premise outlined in A Future of Medicine. I don't think this will be the full story to the totality of medicine. Because, simulations simply do one thing, they simulate, and therein, we build a probable stochastic model from such. Put simply, it's a mechanism that may be used to answer yes or no questions, but of course, what precedes treatment to some illness is the detection of such.

In his book called "The Body: A Guide for Occupants", Bill Bryson regarded cancer and other such

diseases as "system failures", and therein, he remarked, part of the problem is that our nervous system, paradoxically, may not register the early formation of cancer and other such events (given it's severity). But, perhaps we can go even further than the mere detection of cancer.

What precedes treatment is detection, and when this is continuous, we may regard such as monitoring, and this is where we enter into the domain of molecular programming. Can we run computational devices within cells? Yes. Furthermore, as Sarpeshkar noted in his paper, Analog synthetic biology, electronics (simulation) and chemistry (e.g. biochemical computation) are deeply linked. As Sarpeshkar wrote,

> "There are striking similarities between chemical-reaction dynamics (figure 3a) and electronic current flow in the subthreshold regime of transistor operation (figure 3b): electron concentration at the source is analogous to reactant concentration; electron concentration at the drain is analogous to product concentration; forward and reverse current flows in the transistor are analogous to forward and reverse reaction rates in a chemical reaction; the forward and reverse currents in a transistor are exponential in voltage differences at its terminals analogous to reaction rates being exponential in the free-energy differences in a chemical reaction; increases in gate voltage lower energy barriers in a transistor increasing current flow analogous to the effects of enzymes or catalysts in chemical reactions that increase reaction rates; and the stochastics of the Poisson shot noise in subthreshold transistors are analogous to the stochastics of molecular shot noise in reactions. [...] The logarithmic dependence of the electrochemical potential in chemical concentration or of current enables one to map log-domain analog transistor circuit motifs in electronics to log-domain analog molecular circuit motifs in cells and vice versa."

Which he further extended and then summarized to,

> "the cytomorphic [cell derived computation] mapping between electronics and chemistry outlined in [...] enables one to map from electronic circuits to DNA–protein circuits and vice versa. [...] The other direction [...] is useful for the design and simulation of synthetic biological circuits or the ultrafast stochastic simulation of large-scale systems-biology circuits with supercomputing chips."

Therefore, we have an isomorphism between simulation on electronic infrastructure and biochemical reac-

tions.

Which I argue, has far reaching implications. Because this reduces modeling to programming, and from programming, abstraction. Why does this matter? When we abstract, we may sometimes reduce 'complex' things into simpler, more conceptually 'discrete' things. Which therein, due to this simpler nature, may further permit others to effectively build upon such, and therein, may create something more sophisticated, perhaps even, greater than the sum of it's components.

That is, a system where discrete units build upon other discrete units and therein produce more complex non-discrete units. This system permits for abstraction, so complex non-discrete units may be abstracted into simple discrete units. Thereafter this process of production and abstraction enables further production and abstraction and so forth. Each iteration or generation may be considered to be more sophisticated than prior generations, given that each generation is a product of prior generations… From this analogy, you can imagine these bottom-up and cumulative processes will eventually give rise to very sophisticated products, and perhaps one day, akin to how emergence gives rise to the complexity found in nature.

From personal experience, my `https://imager.io` project wouldn't be possible without the various open source components it's built upon. Simply because my time is finite, and especially because lower-level encoding details are just **too complicated for me to understand and implement on my own**. I am nevertheless able to compose such components into a larger and more sophisticated end product, from the preexisting output of resources and information from the global open source, software community. Overall added value that may be considered to be greater than the sum of its components, and therefore emergent in a manner of speaking. In an old English paper I likened the open source community as "the printing press of computable knowledge", and perhaps even more significant than the advent of the printing press itself, because as the industrial revolution introduced a force multiplier of human muscle, so too does abstraction introduce a force multiplier of the human mind.

Because, as I've written, while a book may describe a life's work in mathematics and applications therein, the medium is itself rather passive. A book may describe a life's work in applied mathematics, yet a mind is required to manifest its application. Whereas, imagine a medium where the most knowledgeable of experts can record their understanding of a given domain as functions that map problems to solutions, in a manner that can be utilized by any layperson, and thereafter this record can be reapplied, reused, and so forth,

forever thereafter, and, in this manner, what are the ramifications of such?

If you can build upon abstracted systems written by experts, where these abstractions mask complexity, we may then presume that the overall barrier to entry will drop. That is again, if laypersons are able to built upon abstracted systems in a manner that doesn't require an expert understanding or formal education in such (and presumably in a manner akin to my aforementioned `https://imager.io` story). But on the other end, if the barrier to entry is lower, this implies reduced costs, and therefore, we may likewise see significant cost savings in industries built upon such, perhaps in a manner akin to using "higher level" programming languages for applicable problems.

The field of synthetic biology is one that attempts to unify engineering and biology. For this intersection, we have a foundation that reduces biology to software. With a proper software ecosystem in place, these bottom-up, cumulative processes may give rise to very sophisticated products, and perhaps one day, akin to the emergent phenomena seen in nature itself. Because, such an ecosystem is akin to the aforementioned "force multiplier of the human mind".

But perhaps too, just because software is typically easier to experiment and iterate on. You don't need a lab with specialized knowhow for the equipment, but in this case, perhaps, access to some cloud based compute infrastructure that affords easy and cheap access to the more specialized analog computing hardware. (If I happen to get into this, I'd perhaps call it SubSystems, given my `sub.systems` GTLD.)

What may concern us now, the implementation of analog computers for such workloads, but, as many have pointed out. When we speak of digital vs. analog. What we are really discussing is, computation in terms of discrete vs. continuous analogs.

Digital computation is discreet, and proceeds in terms of discrete steps. All representations in digital form, are mere, and meaningless symbols. Computer arithmetic for instance, is simply the manipulation of such symbols is manner that implements such operations. This is also, the greatest strength to discrete computation. Remember those pen n' paper procedures you were taught for adding arbitrary numbers together? Digital computation implements the very same processes, and so, can scale to adding arbitrary numbers without loss of precision[(Hehner)].

Here, the versatility of discrete computer architectures cannot be understated. For instance, is there utility

in simply simulating analog computation on digital hardware? Furthermore, if the criticism is about the von neumann bottleneck, then such really pertains to the limitations of the von neumann computer architecture, and therefore doesn't disqualify the entire field of discrete computer architectures.

Furthermore, analog architectures are encumbered by a multitude of hurdles.

[...]

Sarpeshkar described the output of an analog computer as a "1 multi-bit" channel, can this be said to be a function? As is, an analog computer, is presumably initialized VIA a 'discrete' input from a digital computer. Therefore, perhaps analog computers could be modeled as a Comonad from Haskell. That is, in Haskell syntax, akin to the following relation, $\forall\ a\ b.\ a\ \ \rightarrow\ \ m\ b$. Perhaps then, translating this to digital form is simply mapping $m\ a \rightarrow a$, therein, loosing context that can only be represented in an analog computer. Furthermore, if we constrained this to $\forall\ a.\ a\ \rightarrow\ m\ a$, then subsequently, all further computation in this machine will proceed as a series of natural transformations of the form $\forall\ a.\ f\ a \rightarrow g\ a$, since a is never transformed, this should ensure continuity of all values of x in a. But overall, in analog computation, perhaps it'd suffice to say that an analog model is parameterized in terms of time and space, and is therefore limited to continuous functions that satisfy the domain.

Digital computers manipulate symbols, and can assign any arbitrary meaning to such. Whereas analog computers must maintain continuity, and therefore, must be defined as a series of transformations. The transformations may be a program, where the transformations themselves are built upon some more primitive set of gates.

We can imagine these transformations as mappings of some $a \rightarrow a$, and where function application is an application between functions.

Each gate may be defined as some $\forall a$. Constraint $a \Rightarrow\ \ a \rightarrow a$, where Constraint a represents some constraint that the composition of these gates must collectively satisfy. This provides some level of formal verification for our software. Given that, for instance, debugging such may be extremely difficult, or perhaps even, impossible. Furthermore, perhaps we can regard the unification of all of such constraints as the set representing the domain of our program, which we will regard as S. But, what about functions that transform the domain? For instance, what if some function f introduces an asymptote, and another function g cancels it

out, and therefore the composition should likewise remove the constraint for subsequent functions. Likewise, addition should shift all the values within the domain by a given amount, including e.g. asymptotes.

[...]

Analog Computer Introduction

Digital computation is discreet, and proceeds in terms of discrete steps. All representations in digital form, are mere, and meaningless symbols. Computer arithmetic for instance, is simply the manipulation of such symbols is manner that implements such operations. This is also, the greatest strength to discrete computation. Remember those pen n' paper procedures you were taught for adding arbitrary numbers together? Digital computation implements the very same processes, and so, can scale to adding arbitrary numbers without loss of precision(Hehner).

Conversely, the central issue plaguing analog computers is that of precision. While digital computers can arbitrary scale the resolution underlying numerical workloads, analog computations are typically bounded to just three or four bits of precision.

Overall, the versatility of discrete computer architectures cannot be understated. For instance, is there utility in simply simulating analog computation on digital hardware? Furthermore, if the criticism is about the von neumann bottleneck, then such really pertains to the limitations of the von neumann computer architecture, and therefore doesn't disqualify the entire field of discrete computer architectures.

Yet, advocates of analog computers argue that many problems do not require high precision computations. Citing for instance, the limitations of real world measurements themselves.(MacLennan) Likewise, Sarpeshkar argues that the fundamental limitations of analog computing, notably, noisy signals, is ideal for simulating stochastic process. Because simulating randomness on digital hardware imposes synchronization constraints that significantly impacts performance on such systems. Whereas on analog architectures, Sarpeshkar argues, you essentially get such for free.

Yet, while discrete computer architectures are incredibly versatile, it may not necessarily be efficient.

and furthermore, many workloads may not require high precision computations and therefore needn't pay

the price in energy consumption.

Furthermore when it comes to solving say, differential equations, analog computing is unrivaled. Consider the following digram that implements $\ddot{y} = -y$.

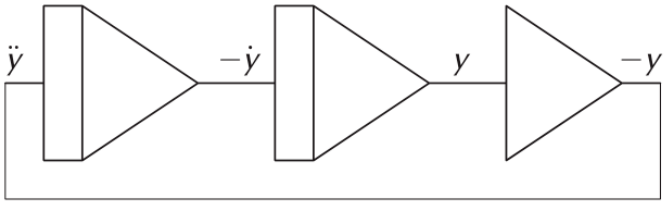

Figure 1: "The basic circuit for solving $\ddot{y} = -y$. From left to right we have two integrators and a summer (with each component inverting the sign)". Source: https://chalkdustmagazine.com/features/analogue-computing-fun-differential-equations/.

As you can see, the leftmost elements are the integrators, while the rightmost element is called the summer.

As MacLennan wrote, in digital computation, quantities are rather arbitrary symbols that have no direct relationship to the physical systems that such may be tasked with simulating. In contrast, generally, analog and physical phenomena are governed and defined by the same mathematical laws. Or as MacLennan put it, "the computational quantities are proportional to the modeled quantities."

Perhaps it goes without saying then, that implementing the same processes in digital computer architectures would require significantly more infrastructure in implementing the same mathematical laws in terms of manipulations of arbitrary symbols. Conversely, there is an elegant economy to the above diagram, and furthermore, it implements such without requiring stored computer memory, and this itself is notable, as I will try to explain.

Imagine we somehow needed to compute one instruction for each atom, for each cell, in the human body. Then we will require $4\text{x}10^{27}$ instructions.

To imagine this in terms of time. Consider first, simply the latency overhead incurred when your processing hardware and main memory is physically separated from each other, and therefore, each instruction must first read some datum from main memory. Lets say this overhead is 100ns per instruction. Then latency overhead alone will amount to $4 \text{ x } 10^{20}$ seconds, or in other words, latency overhead alone will total $12,683,916,800,000$

years per evolution.

But alas, this is a ludicrous exercise for a multitude of reasons. Because, for instance, what is meant by one instruction per atom? For each evolution, can the state of each atom be computed without considering neighboring interactions? But crucially, remember the aforementioned digram that implements $\ddot{y} = -y$? Such problems needn't concern such implementations.

Real World Implementation

References

(Does Thinking Really Hard Burn More Calories?) https://www.scientificamerican.com/article/thinking-hard-calories/

(NVIDIA A100 TENSOR CORE GPU) https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf

(The Thermodynamics of Brains and Computers) https://webhome.phy.duke.edu/~hsg/363/table-images/brain-vs-computer.html

(Bernard) Dr. Bernard, A Future Of Medicine. Heme Review, https://youtu.be/iVt5BpoTHYg.

(Sarpeshkar) Sarpeshkar, Rahul. Analog Supercomputers: From Quantum Atom to Living Body, https://youtu.be/ZycidN_GYo0.

(Hehner) Hehner, Eric & Horspool, R.. (1979). A New Representation of the Rational Numbers for Fast Easy Arithmetic. SIAM J. Comput.. 8. 124-134. 10.1137/0208011.

(O'Donnell) O'Donnell, Kevin. (2018). Digital computer simulation of an electronic analog computer. 1-7. 10.1109/LISAT.2018.8378025.

(Achour) Achour, Sara & Rinard, Martin. (2020). Noise-Aware Dynamical System Compilation for Analog Devices with Legno. 149-166. 10.1145/3373376.3378449.

(The AI Hardware Problem) The AI Hardware Problem, YouTube.