Future Computers: digital, quantum, biological

Industry-standard architecture models for the next generation of general-purpose Computers

Philip Treleaven
University College London

Abstract

Quantum computers offer huge potential performance; and Biological computers (DNA/RNA programmable) can revolutionise pharmaceuticals. However, to exploit these potentials, firstly we need simple 'engineering' descriptions of Quantum computers and Biological computers which seems challenging; and secondly *industry-standard* models (cf. von Neumann model) embedded in the architecture and programming languages.

The contribution of this paper is a) 'laypersons' descriptions of Quantum and Biological computer architectures, b) comparisons with digital computers, and leading to c) discussions of industry-standard models.

1. Future Computers

The digital (von Neumann control flow, stored program) computer model has underpinned technological progress for 60 years [von Neumann, 2020]. However, supporting a sequential program execution architecture, devices are reaching their physical limits in terms of miniaturisation, scalability and performance [Bennett, 2011]. This is driving interest in new computer architectures. Valuable insights can be gained by comparing digital and Quantum computers, also with biological information processing [Bassel, 2018].

The von Neumann model is *the* industry-standard because it is common to control flow computers and procedural languages. What we require are general-purpose, scalable models embedded in computers and associated programming languages:

- **Digital computer** – a parallel variant of the traditional binary, control flow, stored program computer model; multi-instruction-multi-data (MIMD) stream computers.
- **Quantum computer** - information processing using Quantum Bits (Qubit), Quantum entanglement, and 'configurable' Quantum logic gates for processing.
- **Biological computer** – information processing using cells (protein synthesis); DNA, proteins and RNA to create new cells (cf. hardware) or enact computational operators.

The key to progress is firstly simple 'engineering' descriptions of Quantum and Biological computers and secondly common (industry-standard) models embedded in computer architectures and programming languages. A standard architecture for Quantum computers may deliver major computational benefits. In turn, specifying a general-purpose architecture for Biological computers (cf. cell and molecular biology) has the potential to unlock understanding of information processing in nature, and our ability to engineer cells programmed at the DNA/protein level for applications such as immunology.

2. Computation Mechanisms

Computation is defined as the controlled transformation of information, sensitive to the representation's properties; determining the behaviour of information processing [Denning, 2011]. The profound distinction is that natural (biological) computation uses physical structures that directly transform themselves; whereas artificial (human-made) computation uses abstract structures that interpret a model or simulation.

We propose four foundational engineering paradigms underpinning computation:

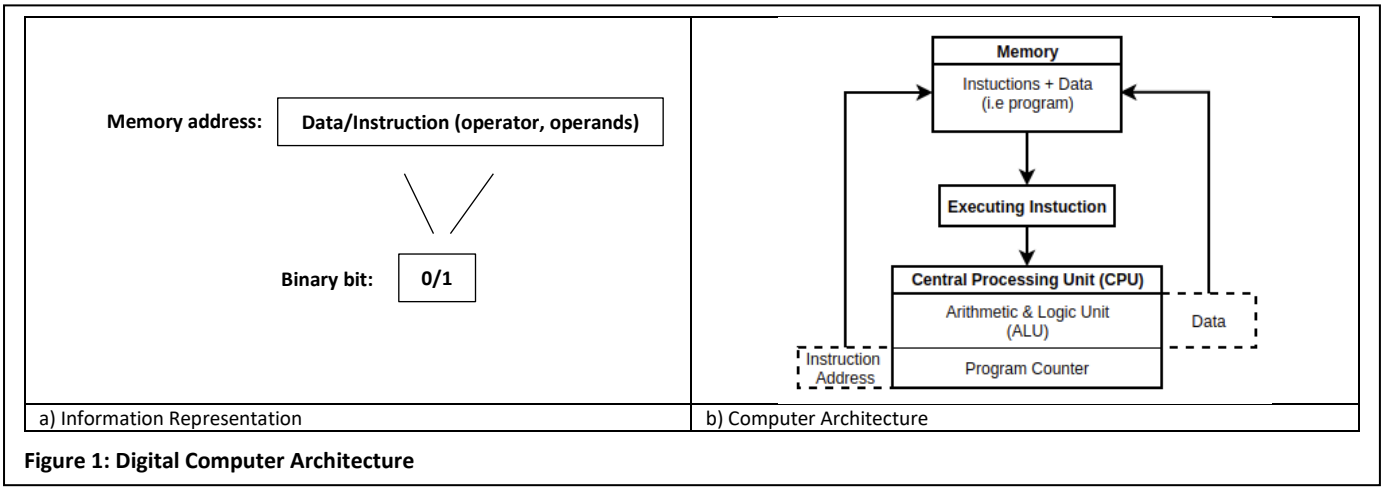
- **Information** - the way information is encoded: a) *discrete/digital* with information represented by a series of values of a physical quantity, such as binary, DNA ATGC; b) *continuous/analog* with information represented by a variable physical quantity, such as Qubit, voltage.
- **Structure/processing** –the way computation is represented and processed: a) *interpreted model* - an abstract model simulated by digital program/data; b) *direct transformed* physical structure, such as DNA/proteins.

- **Control** - the way information is selected for processing: a) *explicit* such as instruction addresses; b) *pattern matching* for example proteins.
- **Communication** - the way information and state changes propagate: a) *multicast* or broadcast, such as shared memory; b) *unicast* or point-to-point, such as messages.

3. Digital Computers

Digital binary, stored program, control flow computers (see Figure 1) comprise an addressable memory containing data and instructions, and central processing unit (CPU) that interprets instructions. Being able to write data and then execute as instructions is a powerful basis for general-purpose computation. The CPU comprises an arithmetic and logic unit (ALU) and a Program Counter defining the memory address of the next instruction(s) to be executed.

In the late 1940's a number of digital stored program computer architectures were proposed but the von Neumann architecture became the industry standard model, embedded in computers and procedural languages. The model's instructions comprise an (ALU or control) operator and operands (data or memory addresses). With an ALU instruction, the Program Counter automatically increments. With a control instruction, the memory address overwrites the Program Counter.

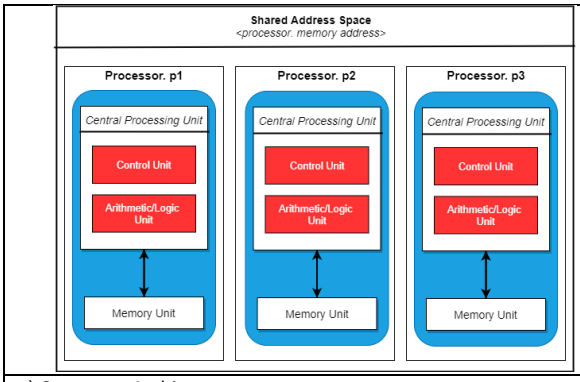


a) Information Representation | b) Computer Architecture

Figure 1: Digital Computer Architecture

As a *benchmark* for our discussion of Quantum and Biological computers, we present a scalable control flow model.

For a general-purpose parallel control flow computer, we need an inherently MIMD architecture (cf. von Neumann) model embeddable in: a) parallel computers; and b) procedural languages (see Figure 2). For instance, each computer's local memory might form part of a shared address space; and control instructions could specify multiple (next) instruction addresses.



a) Computer Architecture | b) Programming Language

Figure 2: Example General-purpose parallel architecture model

Programming model

This parallel computer model can be retrofitted into procedural languages by including the concepts in Figure 2 of a) process (p1, p2); b) shared address space (p1.x, p2.x); c) 'fork' control (|); and d) possibly some form of synchronisation statement.

Next, we provide a simple but broadly accurate 'engineering' description of a Quantum computer.

4. Quantum Computers

Quantum computers, typically gate-based [Quantum Computing, 2021], are analogous to programming at the circuit logic level or configuring a Field Programmable Gate Array (FPGA). We start with some definitions:

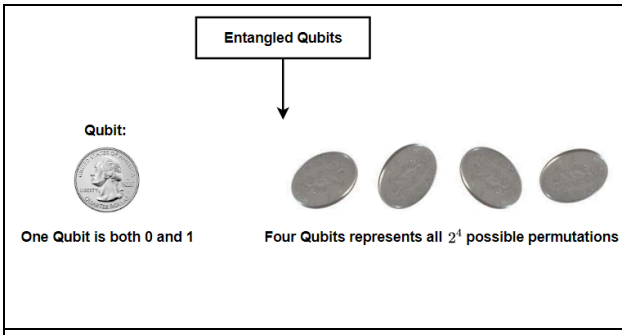
- **Quantum bit** – a Qubit is the basic unit of quantum information, representing simultaneously two values/states (0 and 1).
- **Quantum register** - a Quantum register is a system comprising multiple linked Qubits. It is the Quantum analog of the classical processor register (i.e. Entanglement).
- **Quantum gate** – a Quantum logic gate is a basic quantum operator; a circuit processing a small number of Qubits.
- **Quantum circuit** - a model in which a Quantum computation is a configuration of Quantum gates.
- **Quantum parallelism** – the ability of a Quantum computer to process in parallel all values of a Quantum register in a single computation (i.e. Superposition).
- **Quantum instructions** – specifications for configuring the Quantum gates to perform a Quantum computation.
- **Quantum programming** - the process of assembling sequences of instructions, called Quantum programs, capable of running on a Quantum computer.

Physicists discuss Quantum computers in terms of Quantum states that are specified by probability amplitudes and two continuous variables are needed to uniquely specify the state of a Qubit [Quantum Computing, 2021]. However, it is popular to contrast a Qubit (0 and 1) with binary (0 or 1). A colloquial analogy is to consider information as represented by a 'coin'. Binary information is a coin's head or tail. A Qubit is depicted as a *coin balancing or spinning on its edge* on a on a table; representing both 0 and 1. S*toring* a value in a Qubit is 'spinning' our coin between 0 and 1 (cf. in limbo). *Reading* a Qubit is equivalent to 'banging the table' causing the 'coin' to fall as 0 or 1. Reading 'freezes' both the output and also the stored value (i.e. state) of the Qubit (0 or 1). It also discards information on how the value was represented by the Qubit (e.g. polarity, spin, etc.). Due to a Qubit being able to support two states at the same time: 1-Qubit stores two states (01), 2-Qubits stores four states (00) (01) (10) (11) and 8-Qubits stores 256 states (00000000) … (11111111). Importantly when a value is stored in a Qubit, a probability can be added to influence the output. Continuing our coin analogy, this is like adding a *bias* (cf. biased coin) changing the probability of an outcome.

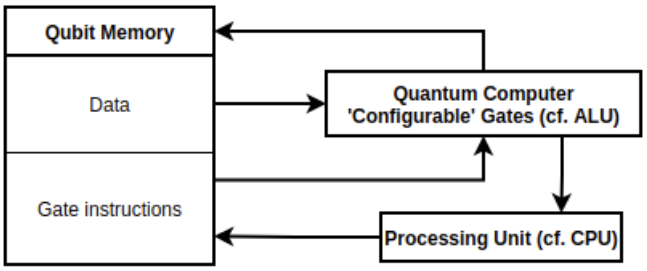
Four key concepts in Quantum computers are: a) *Superposition* – a Qubit represents information as 0 and 1; b) *Entanglement* – Qubits are linked together as a register, linked behaviour; c) *Interference* - controlling Quantum values' probability and amplifying the signals leads towards the right answer; and d) *Coherence/decoherence* – since Qubits lose information over time, estimating the 'shelf-life' of information is important. With Quantum entanglement, a group of Qubit or register is formed so that actions performed on one Qubit affects the others. (Entanglement is created, for example, by microwave pulses or using a laser to split photons into pairs.) With Quantum interference, adding a probability influences the output towards the right result.

Quantum processing is performed by Quantum logic gates configured by Quantum instructions. Qubit data is input, passed through the Quantum gates, producing outputs. Configured gates provide a mapping/transformation function. For a Quantum computer, the inputs to the Quantum gates are (0 or 1), and the outputs to the Quantum memory (0 and 1) modified by a probability.

The simplistic Quantum computer model (Figure 3) follows digital computers with the Qubit memory based on Qubit registers (cf. memory words) containing data and gate instructions in the same memory. The addressing structure for memory access requires a separate Processing Unit to select data and gate instructions for processing using these addresses.



a) Information Representation



b) Computer Architecture

Figure 3: Quantum Computer architecture

Designing a Quantum computer, guided by our computation mechanisms (section 2): a) *Information* - Qubits represent computation; b) *Structure/processing* – uses Quantum entanglement to build structure and Quantum processor interprets the computation; c) *Control* – a Quantum control mechanism might use *explicit* memory addresses; and d) *Communication* – a Qubit memory supports *multicast*.

Quantum computers subdivide into:

- **Quantum hybrid** – current Quantum computers are hybrids comprising a traditional digital computer for program control, together with a Quantum co-processor comprising Qubit memory and Quantum logic gates for execution. The digital computer is used to 'write' the Qubit memory, configure the Quantum gates, and select the desired result.
- **Quantum unitary** – a future general-purpose Quantum computer arguably needs: a) the Qubit memory to store *data* and *gate instructions* (cf. shared memory); b) a Qubit addressing mechanism identifying a specific Qubit; c) a set of universal Quantum gate operators for all programs; d) a processing unit for the gate instructions to configure the Quantum gates (cf. CPU); e) outputs that store a superposition value in the Qubit memory; and f) addressing mechanisms for selecting proceeding gate instructions (cf. Program Counter).

A Quantum computer has three sources of power: a) *Qubit correlations* – information is stored in correlations using entanglement; b) *Qubit parallelism* – a group of Qubits, connected by entanglement and guided by inference, test all combinations of values in parallel (cf. slot machine dials); b) *Gate parallelism* – output flow through a network of Quantum logic gates in parallel.

Programming model

Importantly, a Quantum algorithm and program is independent of the need for Quantum computer hardware. That said, what is required for Quantum computers is a common 'industry-standard' model [Smith et al 2017]. This model then needs to be embedded in a Quantum hardware architecture and corresponding Quantum programming system. Already, Quantum programming systems exist, including Quantum instruction sets, Quantum software development kits (SDK), and Quantum programming languages. Quantum programming languages divide into: a) imperative, such as Quil, QCL, Q# and Silq; and b) functional, such as QFC/QPL and QML [Quantum Programming, 2021].

When designing a Quantum hybrid co-processor many of the engineering challenges (addressing Qubits, configuring Quantum gates, and writing a distribution into a Qubit) can be handled by the digital computer. Exploiting Qubit parallelism (i.e. superposition) for performance using digital technology is less obvious, compared to Quantum gates [Lanzagortaa & Uhlmannb, 2008].

Designing a general-purpose Quantum unitary computer raises a number of interesting questions. Firstly, for *Quantum memory,* how Qubits and Qubit registers represent data, gate instructions and addresses. Secondly, for *Quantum gate instructions,* how Qubit values can specify gate configurations. Thirdly, for *Quantum addressing,* how to access a specific Qubit register and Qubit which may require an address *tuple*. Fourthly, for *Quantum processing,* how are all possible combinations of Entangled Qubits tested. Finally, a 'left-field' solution for Quantum computers may be Optical devices [Optical Computing, 2021] due to non-interference, or Biological devices using a DNA/proteins approach, and pattern matching control.

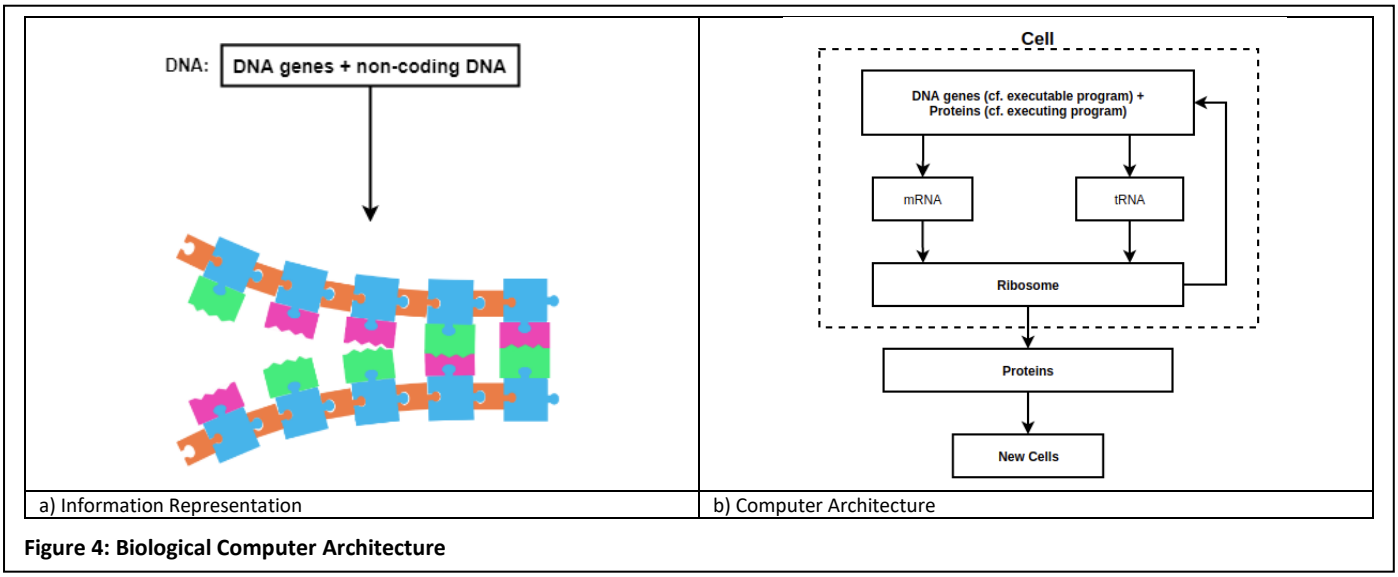
5. Biological Computers

The reassuring fact about biological information processing is that we know it works [Bray, 1995].

The key distinction of Biological over digital computers, as discussed, is that natural information processing transforms the physical structure (cf. hardware) and/or its environment. Figure 4 illustrates biological information processing:

- **DNA** – comprises genes (and non-coding DNA), composed of 4 nucleotide bases adenosine (A), thymine (T), guanine (G), and cytosine (C). Genes contain the information needed to make proteins. DNA is analogous to a stored executable program (cf. instructions + data).
- **RNA** – comprises a copy of DNA constituting a gene, composed of 4 nucleotide bases adenosine (A), guanine (G), cytosine (C) and uracil (U). RNA often acts as the control mechanism, binding to 'complementary' DNA fragments and modifying its activity (regulating gene expression).
- **Proteins** – comprise one or more chains of amino acids, composed of 20 different types of amino acids. Proteins appear analogous to an executing program.

The ATGC bases of DNA, AGCU of RNA and the 20 amino acids of proteins are akin to jigsaw puzzle pieces. Sequence of bases in the DNA determines the sequence of bases in the RNA molecule, which in turn determines the sequence of amino-acids comprising the protein. Information encoded in their sequential structure flows unidirectional from DNA through RNA to proteins (known as the *Central Dogma*). Computation in Biological computers centres on molecular (or molecular assemblies) 3D shape pattern matching and processing.



a) Information Representation | b) Computer Architecture

Figure 4: Biological Computer Architecture

The journey from gene to protein consists of two major steps: a) *transcription* - information stored in a gene's DNA is transferred by messenger RNA (mRNA) that acts as a 'blueprint' for creating proteins; and b) *translation* - a ribosome uses mRNA and transfer RNA (tRNA) to assemble a protein. Processing is driven by powerful pattern matching (control) mechanisms that manipulate the DNA structure.

The flow of information from DNA to RNA to proteins (i.e. protein synthesis) is one of the fundamental principles of biology. Enzymes (i.e. polymerases) assemble DNA and RNA, and ribosomes produce proteins for which they receive mRNAs. An mRNA molecule is (approximately) a copy of a DNA sequence that describes the sequence of amino acids in a chain that (when properly folded) makes up the corresponding protein. A tRNA of a given shape transports one specific amino acid, and attaches to a specific fragment of mRNA – this high degree of specificity of shapes ensures that, in general, only the correct amino acid can be attached to the growing chain.

Design of a general-purpose Biological computer guided by our computation mechanisms (section 2) comprises: a) *Information* – represented by physical quantities (ATGC of DNA, AGCU of RNA, 20 amino acids of proteins); b) *Structure/processing* – direct transformed physical structure (e.g. ribosomes); c) *Control* – pattern matching

mechanism based on the information content (e.g. polymerases); and d) *Communication* – not clear if multicast, unicast or both.

To give a specific example, the pioneering work on mRNA therapeutics for vaccines is likely to drive Biological computer development. An RNA vaccine consists of an mRNA strand that codes for a disease (i.e. specific antigen). Once the mRNA strand in the vaccine is inside the body's cells, the cells use the genetic information to produce the antigen. To produce an mRNA vaccine, scientists use a synthetic equivalent of the mRNA that a virus uses to build its infectious proteins. The cells read the mRNA as 'instructions' to build that antigen protein, and the immune system detects these viral proteins and starts to produce a defensive response to them. Fast forward to 'general-purpose' mRNA therapeutics.

An excellent illustration of Biological 'machine code' is the 4284-character Covid BNT162b2 mRNA code [Berthub, 2020], shown in Figure 5. This code is uploaded to a DNA printer that converts the digital characters to actual biological DNA, and then RNA. Once inside a cell, the RNA is used to produce proteins typical for the virus, which prompts the immune system to develop defences against it. The machine code in Figure 5 exploits the existing DNA and protein 'program' in the cell, which in turn leads to activation of other programs in nearby immune cells.

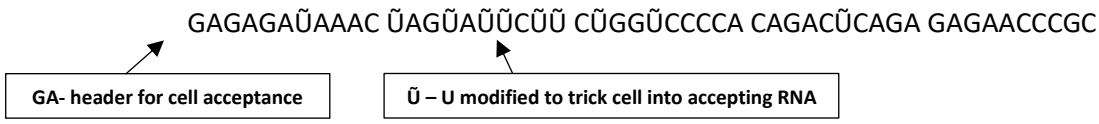


Figure 5: Covid BNT162b2 mRNA 'machine code' (1st 50 characters)

Programming model

As discussed, further progress towards a general-purpose Biological computer and programming language, requires a standard model where we can specify DNA, proteins, RNA strings, and possibly a template library of customisable DNA and proteins. In terms of the standard model: a) 'data/instructions' comprise DNA, RNA and proteins; b) 'operators' support *transcription* and *translation*; and c) a 'control mechanism' based on pattern matching controls execution (e.g. gene editing CASP/CRISP).

Important questions arise: firstly, what chemicals (cf. 'hardware') exists to create a Biological computer; secondly, how the pattern matching control mechanism might work; and thirdly what will a high-level programming language look like.

6. Conclusions

As discussed, *the* industry-standard von Neumann model has underpinned technological progress, but is reaching its physical limits. This has spurred massive interest in Quantum computers. Likewise, understanding information processing in natural systems such as cells and plants can lead to a breakthrough in (DNA) programmable Biological computers [Kari, 2008; Denning, 2011]. (We should also re-visit multi-instruction-multi-data stream digital parallel computers.)

Using our four computational mechanisms discussed in section 2, Figure 6 presents a summary comparison of digital, Quantum and Biological computer models.

	Digital	Quantum	Biological
Information	*discrete/digital* with information represented by physical quantity (e.g. binary)	*continuous/analog* with information represented by a variable physical quantity (e.g. Qubit)	*discrete/digital* with information represented by physical quantity (e.g. DNA bases ATGC)
Structure/processing	*model* - an abstract model simulated by program/data and interpreted.	*model* - an abstract model simulated by program/data and interpreted.	*direct* structure transformed.
Control	*explicit* such as instruction addresses	*explicit* probably using addresses	*pattern matching*
Communication	*multicast* or broadcast	*multicast* or broadcast	*multicast* and/or *unicast*

Figure 6: Comparison of Digital, Quantum, Biological computers

In conclusion, general-purpose scalable architecture models for digital, Quantum and Biological computers can provide powerful new systems. More importantly, this research should contribute to understanding Biological information processing [Mitchell, 2011]. The starting point is simple 'engineering' descriptions of Quantum and Biological computers.

7. Acknowledgements

Dr Martin Schoernig is acknowledged for extensive discussions and contributions to this paper. Prof Stephen Emmott, Dr Alexei Kondratyev, Dr Cat Mora, and Sarvesh Rajdev are thanked for reviewing numerous drafts of this paper.

8. References

[Bassel, 2018] Bassel, G., Information Processing and Distributed Computation in Plant Organs, Trends in Plant Science, vol. 23, no. 11, 2018, www.cell.com/trends/plant-science/fulltext/S1360-1385(18)30184-5

[Bennett, 2011] , Bennett, C., Landauer, R., The Fundamental Physical Limits of Computation, Scientific American, June 2011, www.scientificamerican.com/article/the-fundamental-physical-limits-of-computation/

[Berthub, 2020] Reverse Engineering the source code of the BioNTech/Pfizer SARS-CoV-2 Vaccine, https://berthub.eu/articles/posts/reverse-engineering-source-code-of-the-biontech-pfizer-vaccine/

[Bray, 1995] Bray, D., Protein molecules as computational elements in living systems, Nature, vol. 27, 1995.

[Denning, 2011] Ubiquity Symposia, https://ubiquity.acm.org/symposia2011.cfm?volume=2011

[Kari, 2008] Kari, L., Rozenberg, G., The Many Facets of Natural Computing, CACM vol. 51, no. 10, Oct 2008.

[Lanzagortaa & Uhlmannb, 2008] Lanzagortaa, M., and Uhlmannb, J., Is Quantum Parallelism Real?, https://www.researchgate.net/publication/252477910_Is_quantum_parallelism_real

[Mitchell, 2011] Mitchell, M., Ubiquity symposium: Biological Computation, Ubiquity, 2011, 3, http://delivery.acm.org/10.1145/1950000/1944826/a1-mitchell.pdf?ip=128.16.12.209&id=1944826&acc=OPEN&key=BF07A2EE685417C5%2ED93309013A15C57B%2E4D4702B0C3E38B35%2E6D218144511F3437&__acm__=1546851025_35a94fcfcdef910b6f064e1ad8b0b5a7

[Optical Computing, 2021] Optical computing, Wikipedia, https://en.wikipedia.org/wiki/Optical_computing

[Smith et al, 2017] Smith, R., Curtis, M., Zeng, W., A Practical Quantum Instruction Set Architecture, https://arxiv.org/abs/1608.03355

[Quantum Computing, 2021] Quantum Computing, Wikipedea, https://en.wikipedia.org/wiki/Quantum_computing

[Quantum Programming, 2021] Quantum Programming, Wikipedea, https://en.wikipedia.org/wiki/Quantum_programming

[von Neumann, 2020] von Neumann Computer Architecture, Wikipedia, https://en.wikipedia.org/wiki/Von_Neumann_architecture