Impact of Emerging Computing Architectures and Opportunities for Process Systems Engineering Applications

David E. Bernal [a,d], Carl D. Laird[b,1], Stuart M. Harwood [c], Dimitar Trenev[c], and Davide Venturelli [a,d]

[a] Research Institute for Advanced Computer Science, Universities Space Research Association, Mountain View, CA

[b] Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA

[c] ExxonMobil Technology and Engineering Company - Research, Clinton, NJ

[d] Quantum Artificial Intelligence Laboratory (QuAIL), NASA Ames Research Center, Moffett Field, CA

Abstract

Moore's "law" was the observation that the number of transistors in an integrated circuit doubled approximately every two years. This trend has distinctly failed to hold in recent years. The death of Moore's law has left researchers and practitioners in the computational sciences searching for technologies to provide the speedups formerly supported by Moore's law. Previously overlooked chip architectures and other computing technologies are now receiving more development resources. Critically, these technologies are gaining more mature software support, opening their adoption by researchers in algorithms and applications. In this article, we review some of these computing technologies, their relationship with various algorithms and applications, and their potential benefits (or pitfalls). We close with recommendations for future work by the process systems engineering community specifically.

Keywords

Emerging hardware, distributed computing, quantum computing, analog computing, high-performance computing

1 Introduction

A key tenet of the field of Process Systems Engineering (PSE) is the formal mathematical description of a problem to enable efficient numerical solutions with the aid of a computer. With this, the limits of what is possible in PSE are, of course, linked to what is possible in the area of computing. Problems encountered in PSE require the solution of challenging computational tasks, which require the solution of primordial computational operations such as matrix-vector multiplication, solution of systems of (partial) differential equations, derivatives computation and evaluation, search, and optimization methods. These operations build up to constitute problems relevant to process, chemical, energy, environmental, and systems engineering. Work in PSE has included developing and improving numerical algorithms (largely serial). Steady hardware performance improvements coupled with these algorithm developments have led to repeated successes in solving previously intractable problems in the area of PSE. However, as single core performance improvements slowed, computing hardware breakthroughs now focus on emerging technologies with new capabilities, limitations, and computing paradigms. To see continued performance improvement and innovation, scientific computing research is focused on developing new understanding, implementations, and algorithms that can effectively exploit these emerging computational architectures.

Computational complexity theory captures the ability of an algorithm to scale with problem size and limits the number of steps an algorithm must take to solve a given problem. However, there is still much flexibility in implementing those algorithms. While emerging architectures have the potential for transformative computational performance, they also bring implementation constraints; different architectures have different strengths and weaknesses concerning execution time and power requirements. Using the proper computing hardware for the right job provides opportunities to achieve practical time or energy savings; these savings could make the difference between a problem being "tractable" or not at an application scale. Furthermore, there is significant scope for designing and implementing new algorithms that can take advantage of emerging computational architectures and, in some cases, even co-design the algorithm and hardware simultaneously to improve computational performance significantly [23].

In this article, we provide an overview of some emerging computational architectures, discuss their capabilities and the maturity of software tools, and provide context for these architectures with respect to different algorithms and applications within PSE. We close with some discussion of the maturity of

[1] Corresponding author. Email: `claird@andrew.cmu.edu`.

and applicability of these architectures with recommendations for future work by the PSE community specifically.

2 Emerging Technologies and Their Applications

2.1 Multi-core, Distributed, and Hybrid Parallel Architectures

The early to mid-2000s saw a stagnation in the year-over-year increase in CPU clock speeds, and chip manufacturers focused instead on hyperthreading and the development of multicore architectures to drive performance improvements [58]. Similarly, we also saw a significant rise in the availability of distributed computing clusters for both academic and industrial users that promised scalable parallel computing resources. These changes had a major impact on the landscape of scientific computing today, where parallel computation is now mainstream.

Almost every standard desktop or laptop sold today contains multiple computing cores. Typical multicore systems are affordable, and there is a range of mature, standardized tools for implementing parallel scientific computing codes. While communication between threads can be very fast on these shared-memory architectures, they still typically contain a relatively low number of cores, and for large-scale applications, key bottlenecks include the available bandwidth for "off-chip" memory [22]. Distributed computing clusters, on the other hand, bring a large number of cores by connecting many computational nodes with standard or specialized networking technology. While these architectures can overcome memory bottlenecks by distributing the workload over multiple nodes, communication across the network must be carefully managed for scalable performance.

Graphics processing units (GPUs) can hardly be considered "emerging" hardware anymore, but their impact on various scientific computing problems cannot be overstated. Originally driven by computer graphics requirements, GPUs have become a highly parallel computing architecture that can cost-effectively deliver many operations per second for suitably parallelizable applications, such as dense linear algebra as commonly found in neural networks (NN) training. While they promise massive parallelism at a relatively low cost, these "streaming" architectures come with significant implementation constraints over general CPU-based architectures, and applications must be selected carefully.

Modern distributed computing clusters are hybrid architectures that combine many multicore computing nodes and often include specialized accelerators. Effective use of these hybrid architectures is a major theme of the DOE Exascale Computing Project [5, 24]. The tools for building parallel applications with these architectures (e.g., MPI [28]) are very mature with well-established standards and implementations [27]. Even high-level languages like Python and Julia have mature libraries and interfaces for implementing parallel codes on both shared- and distributed-memory architectures [17, 18, 13]. Extensions built on these packages have enabled scalable parallel optimization implementations for specific applications like large-scale nonlinear programming [47, 34, 62, 53]. Maturing APIs and software support for GPUs (e.g., [36]) has enabled the use of GPUs for a number of scientific computing applications [42], mainly focused on training deep NNs [37] but also including nonlinear optimization [14]. Numerous examples within PSE demonstrate solutions to previously intractable problems through effective use of these architectures.

2.2 Application Specific Integrated Circuits, Tensor Processing Units, and Field Programmable Gate Arrays

The categorization of a device as an application-specific integrated circuit (ASIC) can vary. For this discussion, we consider an ASIC to be a device where an algorithm that runs on it has a significant portion programmed directly into the chip's architecture and thus is fixed at the time of manufacture. D.E. Shaw Research's development of Anton, the supercomputer for performing molecular dynamics simulations, provides a good example of the types of considerations that go into the development of ASICs for a scientific problem [50]. By definition, an ASIC is almost inextricably linked with a particular algorithm or family of computational kernels. This means that there must be mature algorithms for solving the target problem that are unlikely to change over the intended lifespan of the chip. Further, chip design takes time (and money!), and the Anton development team had to consider whether more conventional hardware would advance to the required level of performance in the time it would take them to develop and manufacture the chip. Even with the death of Moore's Law, the rapid progress of GPUs, driven by applications in machine learning, might provide a more cost-effective solution. However, the performance improvements from an ASIC can be huge; the second generation of Anton was over two orders of magnitude faster than conventional HPC and GPUs [51]. Anton may be used for non-commercial research through the Pittsburgh Supercomputing Center.

Another case study is that of the Tensor Processing Unit (TPU). Google saw enough room for improvement over GPUs to develop their custom chip, the TPU [33]. Once again, a careful analysis of alternative technologies and the total costs of ownership was necessary. The main takeaway from these case studies is that while a custom chip is almost certainly faster or more power efficient, the overall economics of the hardware development, purchase, and operation must be considered.

On the other hand, field programmable gate arrays (FPGAs) provide a more flexible alternative to ASICs. Roughly, an FPGA is an integrated circuit with reconfigurable interconnections between the elements. FPGAs are often used for prototyping and testing ASIC design. Like ASICs, FPGAs allow more control of data flow, data reuse, and other optimizations like limited precision fixed-point numbers where possible [52]. Functionally, FPGAs fill a role and have development challenges somewhere between those of GPUs and ASICs. While the software is improving, programming an FPGA is generally not as simple as a GPU. Further, FPGAs tend to have slower clock rates and more limited on-chip memory than GPUs. In an extensive review [52], a general trend was that FPGA-based accelerators for NN inference are more energy efficient than GPU or CPU alternatives but not generally faster. However, with fine control over data and logic flow, FPGAs do permit effective implementation of parallel algorithms; in a review of hardware accelerators for MPC, suitably parallel algorithms saw a modest ($\sim$ 2x) speedup on FPGAs versus CPUs [2].

2.3 Non-von Neumann architectures

Compute-in-Memory refers to techniques that aim to circumvent the bottlenecks in traditional von Neumann architectures – namely, the time and energy bottleneck of data movement through the various levels of memory and onto and off processing units [49]. A common feature of these devices is the ability to do analog matrix-vector multiplication. Fast matrix-vector multiplication enables a number of scientific computing applications, including equation solving, optimization, and machine learning. However, while these devices can perform this operation quickly, their analog nature limits precision. Consequently, compute-in-memory does not suit every application, and taking advantage of it might require hybrid strategies or a fundamental reformulation of the problem. Sebastian et al. [49] review some successful applications of compute-in-memory, including inference in deep NN and iterative linear algebra solvers.

A related idea is that of neuromorphic computing. *Neuromorphic computing* is a field that aims to develop neurologically-inspired computing devices [63, 20]. While many research devices may be called "neuromorphic chips',' the most high-profile examples (IBM's TrueNorth chip [40] and Intel's Loihi chip [20]) focus on efficient implementation of spiking NNs, a particular type of artificial NN that encodes data through the timing of spikes or pulses [61]. As with compute-in-memory devices, a benefit of these chips is their incredibly low power consumption compared to convolutional or other deep NN architectures (potentially 1000 times less for particular devices and problems [20]). Spiking NNs, and thus neuromorphic chips, may be applied to several problems, including various machine learning problems, but also graph search and stochastic optimization [20]. The specific benefits of spiking NNs over other solution methods are unclear, but the low power consumption of neuromorphic chips expands where these problems may be solved to include autonomous or "edge" devices, where power consumption is a constraint. Due to the departure from von Neumann architecture in neuromorphic chip design, proponents of the technology prefer to distinguish between neuromorphic chips and, for example, accelerators for deep learning.

Dataflow architectures are another alternative to the von Neumann architecture. Initially proposed in the 1960s and 1970s as a computing paradigm optimized for data-driven parallel computation [60], academic research on dataflow architectures stalled in the 1980s. However, the rise of deep NN-driven machine learning has motivated the development of commercial systems incorporating ideas from dataflow architectures. Emani et al. [26] have tested one of these systems on scientific applications of deep learning with positive results.

2.4 Physical Annealing and Analog Computing

Historically, the term analog computing was used to refer to computing with physical systems whose evolution mimics the system they were intended to model and simulate. Today that definition has shifted to refer to devices working on the continuum [12]. As opposed to digital computers, in which information is processed in discrete form (and input, output, and intermediate calculations are discretized), analog computers represent variables continuously using various physical quantities (e.g., electrical, mechanical, hydraulic signals) as analogs for the information being processed.

Analog computing devices were widely used throughout history to perform specific calculations, from the ancient-greek Antikythera mechanism used to predict astronomic positions of sky bodies, through the slide rule for computing logarithms, to the advanced military targeting systems that are still in use on navy ships all over the world. Such devices took the back seat after the invention of digital computing machines and the rapid evolution of these computers due to their general applicability and programmability.

As high-performing digital computing systems become more challenging to design, and their increased energy demand makes them expensive to use, analog (or hybrid digital-analog) devices are again a topic of interest due to their speed and efficiency.

One of the more promising examples of this renewed research interest is the physical annealing machines [41], such as the D-Wave quantum annealer or the Coherent Ising machines developed at NTT and Stanford University [32]. These machines attempt to exploit the device's underlying physics to approximate ground solutions to the Ising model, an NP-hard problem equivalent to quadratic unconstrained binary optimization (or QUBO). As many combinatorial and graph-theoretical problems can be reformulated as QUBOs, the potential of physical annealing machines to accelerate the finding of solutions to hard optimization problems is attractive. Nevertheless, there are a number of technical challenges that need to be addressed first. On the engineering side, these include scaling the number of variables and ensuring that the system's connectivity and resolution sufficiently represent the problem with enough precision.

In addition, most practical optimization problems are not only unconstrained and discrete; many problems involve complex constraints and continuous decision variables. While such problems can still be reformulated into QUBOs through discretization and incorporating the constraints using additional variables and penalty terms, the satisfaction of the constraints may only be guaranteed by finding the QUBO's optimal solution. Inexact approximations of the solution might be very close to optimal in terms of minimizing the objective of the problem but still lead to infeasible answers due to the inability to satisfy a specific hard constraint. Nevertheless, the potential availability of an efficient close-to-optimal QUBO solver opens the way for novel algorithms and heuristics that may accelerate the solution of at least some challenging and relevant optimization problems. Considerable effort has been made in developing high-level interfaces to QUBO-based programming, for instance, with the Python-based open-source package from D-Wave Ocean, among several others [46]. Research in overcoming the challenges and defining the practical applicability of the physical annealing machines is ongoing.

Recently, analog mechanisms that perform optimization have also been implemented in neutral-atom devices, where spin variables are represented by atomic qubits trapped in arbitrary 3D configurations via optical tweezers. When operated as annealers [25], the coherence of these systems is considerably larger than flux-qubits in superconducting architectures; however, there are programmability limitations, with early-stage opensource projects supporting their primitives [54]. The current size of devices supports only hundreds of variables, with thousands within reach. These devices are natively implementing interactions that naturally map into Maximum-Independent-Set (MIS) constraints, with compilation techniques similar to minor-embedding required in superconducting annealers [35]. Applications of MIS include scheduling, asset allocation, telecommunication decoding [19], and its analog mode can be used to define quantum sampling protocols with generic machine learning kernel-based applications [31].

2.5 Digital Quantum Computing

Quantum computing (QC) refers to computing and information processing that leverages phenomena explained through quantum mechanics, such as quantum interference and superposition. This computational paradigm does not expand what is computable using non-quantum (or classical) computation, but its promise is that it can accelerate (even exponentially) certain computational tasks [9, 45]. Several computational models can fall under the definition of QC presented above, e.g., adiabatic QC, measurement-based QC, and the quantum circuit model. This section will focus on the quantum circuit model of QC, where algorithms can be implemented as quantum mechanical manipulations of the primary processing unit in QC, the quantum bit or *qubit*, through a set of operators known as gates, which compose what is known as a *quantum circuit*. These (quantum) algorithms can be proved to provide speedups in specific computational tasks compared to algorithms implemented in the non-quantum (classical) setting, and there have been successful experimental realizations of such computational tasks, demonstrating the physical possibility of *quantum supremacy* [9]. The apparent parallelism coming from the seamless operation of information that grows exponentially with respect to the number of qubits in the quantum circuits, together with probability amplitudes that interfere constructively and destructively, are the main ingredients that can explain the theoretical advantage of quantum algorithms. Some of those algorithms with proven advantages are aimed to tackle the simulation of quantum systems, number factorization, and search and optimization [11]. There is a considerable drive to keep developing quantum algorithms, hardware, and software that allows the practical use of this technology in science and engineering applications, with a particular interest in finding those applications where the potential speedups provided by QC can be exploited. In particular, complete software stacks are developed by different companies, providing high-level access to quantum computing simulators and devices, such as IBM Qiskit [7] and Google Circ [21].

Specialized devices known as quantum computers need to be built to implement quantum algorithms, with the stringent requirement of maintaining the delicate quantum state of the qubits while controllably applying the predefined gates. Current quantum computers can reliably implement algorithms for enough qubits (around 50) and gates for their classical simulation to be impractical but too few to implement error correction schemes that can prevent unintended perturbations of the quantum states. These devices have been named noisy intermediate-scale quantum (NISQ) computers. Although the practical realization of most algorithms that provide quantum advantage requires implementing circuits beyond the current capabilities of NISQ devices, one can still leverage the capabilities of these computers to represent probability distributions that are difficult to represent using classical computers. This is mainly done by proposing parameterized quantum circuits integrated into a computational loop with a classical computer to optimize a performance metric of the circuit, in an approach known as variational quantum algorithms (VQA) [16]. This setting is similar to the training of an artificial NN, where the parameterized circuit can be executed by specialized hardware, e.g., a GPU in the NN case or a gate-based quantum computer in the case of VQA. These approaches can still provide quantum advantage using NISQ devices, and algorithms for quantum systems simulation, optimization, and machine learning using this paradigm have been proposed and implemented in the existing hardware [9, 16].

3 Opportunities for the PSE Community

Table 1 summarizes the intersection of emerging computing technologies with disciplines of the process systems engineering community. Emerging technologies have already been successfully applied within several domains, and reasonably mature examples or implementations exist. In other cases, the area may not be mature; however, there is enough preliminary research to indicate potential for further adoption. In this table, we provide some examples of work in each of these areas; however, in the interest of space, this is not an exhaustive list.

These emerging computing technologies offer opportunities to re-think our problem formulations and go-to solution methods. Given the PSE community's adoption of computing technology so far, it is likely that many of these new hardware platforms will become standard tools.

Acknowledgements

DB and DV acknowledge the support of USRA NASA Academic Mission Services (contract NNA16BD14C).

References

[1] Martín Abadi et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Karam M Abughalieh and Shadi G Alawneh. A survey of parallel implementations for model predictive control. *IEEE Access*, 7:34348–34360, 2019.

[3] Akshay Ajagekar and Fengqi You. Quantum computing assisted deep learning for fault detection and diagnosis in indus-

Table 1: Intersection of emerging computing technologies and PSE application spaces. The table includes example references in each area. A (P) indicates areas which we believe are promising for research, and an (M) indicates relatively mature areas.

	Design	Control	Optimization	Data Analytics	Simulation
Distributed / Multicore	[65]	[2]	P [48, 8, 44, 34]	M [1]	M [29]
GPU	P [39]	M [2]	P [14, 55]	M [1, 55]	M [55, 59, 64, 57]
ASICS/FPGA	P [43]	M [38, 2], P [43]	[10]	M [52]	P [51]
Physical Annealing	P [6]	P [6]	P [4, 41, 6]	P [3, 41, 6]	P [6]
Quantum	P [11]	P [15]	P [11, 30]	P [11]	P [11, 56]

trial process systems. *Computers & Chemical Engineering*, 143:107119, 2020.

[4] Akshay Ajagekar, Travis Humble, and Fengqi You. Quantum computing based hybrid solution strategies for large-scale discrete-continuous optimization problems. *Computers & Chemical Engineering*, 132:106630, 2020.

[5] F. Alexander et al. Exascale applications: Skin in the game. *Phil. Trans. of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166), 2020. doi: https://doi.org/10.1098/rsta.2019.0056.

[6] Martin P Andersson, Mark N Jones, Kurt V Mikkelsen, Fengqi You, and Seyed Soheil Mansouri. Quantum computing for chemical and biomolecular product design. *Current Opinion in Chemical Engineering*, 36:100754, 2022.

[7] MD Sajid Anis et al. Qiskit: An Open-source Framework for Quantum Computing, 2021. URL `https://qiskit.org`.

[8] Ignacio Aravena et al. Recent Developments in Security-Constrained AC Optimal Power Flow: Overview of Challenge 1 in the ARPA-E Grid Optimization Competition. *arXiv preprint arXiv:2206.07843*, 2022.

[9] Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[10] Samuel Bayliss, George A Constantinides, Wayne Luk, et al. An fpga implementation of the simplex algorithm. In *2006 IEEE international conference on field programmable technology*, pages 49–56. IEEE, 2006.

[11] David E Bernal, Akshay Ajagekar, Stuart M Harwood, Spencer T Stober, Dimitar Trenev, and Fengqi You. Perspectives of quantum computing for chemical engineering. *AIChE Journal*, 68(6):e17651, 2022.

[12] Olivier Bournez and Amaury Pouly. A survey on analog models of computation. In *Handbook of Computability and Complexity in Analysis*, pages 173–226. Springer, 2021.

[13] Simon Byrne, Lucas C Wilcox, and Valentin Churavy. MPI. jl: Julia bindings for the Message Passing Interface. In *Proceedings of the JuliaCon Conferences*, volume 1, page 68, 2021.

[14] Yankai Cao, Arpan Seth, and Carl D Laird. An augmented Lagrangian interior-point approach for large-scale NLP problems on graphics processing units. *Computers & Chemical Engineering*, 85:76–83, 2016.

[15] Davide Castaldo, Marta Rosa, and Stefano Corni. Quantum optimal control with quantum computers: A hybrid algorithm featuring machine learning optimization. *Physical Review A*, 103 (2):022613, 2021.

[16] Marco Cerezo et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[17] Lisandro Dalcin and Yao-Lung L Fang. mpi4py: Status update after 12 years of development. *Computing in Science & Engineering*, 23(4):47–54, 2021.

[18] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. MPI for Python. *Journal of Parallel and Distributed Computing*, 65(9): 1108–1115, 2005.

[19] Constantin Dalyac, Loïc Henriet, Emmanuel Jeandel, Wolfgang Lechner, Simon Perdrix, Marc Porcheron, and Margarita Veshchezerova. Qualifying quantum approaches for hard industrial optimization problems. A case study in the field of smart-charging of electric vehicles. *EPJ Quantum Technology*, 8(1): 12, 2021.

[20] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. Advancing neuromorphic computing with Loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021.

[21] Cirq Developers. Cirq, April 2022. URL `https://quantumai.google/cirq`. See full list of authors on Github: https://github .com/quantumlib/Cirq/graphs/contributors.

[22] Jeff Diamond, Martin Burtscher, John D McCalpin, Byoung-Do Kim, Stephen W Keckler, and James C Browne. Evaluation and optimization of multicore performance bottlenecks in supercomputing applications. In *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*, pages 32–43. Ieee, 2011.

[23] Sudip S Dosanjh et al. Exascale design space exploration and co-design. *Future Generation Computer Systems*, 30:46–58, 2014.

[24] A. Dubey, L. C. McInnes, R. Thakur, E. W. Draeger, T. Evans, T. C. Germann, and W. E. Hart. Performance Portability in the Exascale Computing Project: Exploration Through a Panel Series. *Computing in Science and Engineering*, 2021.

[25] Sepehr Ebadi et al. Quantum optimization of maximum independent set using Rydberg atom arrays. *Science*, page eabo6587, 2022.

[26] Murali Emani et al. Accelerating scientific applications with Sambanova reconfigurable dataflow architecture. *Computing in Science & Engineering*, 23(2):114–119, 2021.

[27] Thomas M. Evans, Andrew Siegel, Erik W. Draeger, Jack Deslippe, Marianne M. Francois, Timothy C. Germann, William E. Hart, and Daniel F. Martin. A survey of software implementations used by application codes in the Exascale Computing Project. *International Journal on High Performance Computing*, 2021.

[28] William Gropp, William D Gropp, Ewing Lusk, Anthony Skjellum, and Argonne Distinguished Fellow Emeritus Ewing Lusk. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[29] Rudiyanto Gunawan, Irene Fusman, and Richard D Braatz. Parallel high-resolution finite volume simulation of particulate processes. *AIChE Journal*, 54(6):1449–1458, 2008.

[30] Stuart Harwood, Claudio Gambella, Dimitar Trenev, Andrea Simonetto, David Bernal, and Donny Greenberg. Formulating and solving routing problems on quantum computers. *IEEE Transactions on Quantum Engineering*, 2:1–17, 2021.

[31] Louis-Paul Henry, Slimane Thabet, Constantin Dalyac, and Loïc Henriet. Quantum evolution kernel: Machine learning on graphs with programmable arrays of qubits. *Physical Review A*, 104(3):032416, 2021.

[32] Toshimori Honjo et al. 100,000-spin coherent Ising machine. *Science advances*, 7(40):eabh0952, 2021.

[33] Norman P Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[34] Jia Kang, Yankai Cao, Daniel P Word, and Carl D Laird. An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition. *Computers & Chemical Engineering*, 71:563–573, 2014.

[35] Minhyuk Kim, Kangheun Kim, Jaeyong Hwang, Eun-Gook Moon, and Jaewook Ahn. Rydberg quantum wires for maximum independent set problems. *Nature Physics*, pages 1–5, 2022.

[36] Andreas Klöckner. PyCUDA: Even simpler GPU programming with Python. *Nvidia GTC*, 2010.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[38] KV Ling, SP Yue, and JM Maciejowski. A FPGA implementation of model predictive control. In *American control conference*, page 6. Citeseer, 2006.

[39] Yannan Ma, Xi Chen, and Lorenz T Biegler. Monte-carlo-simulation-based optimization for copolymerization processes with embedded chemical composition distribution. *Computers & Chemical Engineering*, 109:261–275, 2018.

[40] Paul A Merolla et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[41] Naeimeh Mohseni, Peter L McMahon, and Tim Byrnes. Ising machines as hardware solvers of combinatorial optimization problems. *Nature Reviews Physics*, 4(6):363–379, 2022.

[42] Manolis Papadrakakis, George Stavroulakis, and Alexander Karatarakis. A new era in scientific computing: Domain decomposition methods in hybrid CPU–GPU architectures. *Computer Methods in Applied Mechanics and Engineering*, 200(13-16): 1490–1508, 2011.

[43] Iosif Pappas, Dustin Kenefake, Baris Burnak, Styliani Avraamidou, Hari S Ganesh, Justin Katz, Nikolaos A Diangelakis, and Efstratios N Pistikopoulos. Multiparametric programming in process systems engineering: Recent developments and path forward. *Frontiers in Chemical Engineering*, 2:620168, 2021.

[44] Cosmin G Petra and Ignacio Aravena. Solving realistic security-constrained optimal power flow problems. *arXiv preprint arXiv:2110.01669*, 2021.

[45] John Preskill. Quantum computing 40 years later. *arXiv preprint arXiv:2106.10522*, 2021. To appear in Feynman Lectures on Computation, 2nd edition, published by Taylor & Francis Group, edited by Anthony J. G. Hey.

[46] Abraham P Punnen. *The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications*. Springer Nature, 2022.

[47] J Rodriguez, Robert Parker, C Laird, Bethany Nicholson, J Siirola, and Michael Bynum. Scalable Parallel Nonlinear Optimization with PyNumero and Parapint. *Preprint at http://www. optimization-online. org/DB HTML/2021/09/8596. html*, 2021.

[48] Michel Schanen, François Gilbert, Cosmin G Petra, and Mihai Anitescu. Toward multiperiod AC-based contingency constrained optimal power flow at large scale. In *2018 Power Systems Computation Conference (PSCC)*, pages 1–7. Ieee, 2018.

[49] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature nanotechnology*, 15(7):529–544, 2020.

[50] David E Shaw et al. Anton, a special-purpose machine for molecular dynamics simulation. *Communications of the ACM*, 51(7):91–97, 2008.

[51] David E Shaw et al. Anton 2: raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 41–53. Ieee, 2014.

[52] Ahmad Shawahna, Sadiq M Sait, and Aiman El-Maleh. FPGA-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, 7:7823–7859, 2018.

[53] Sungho Shin, Carleton Coffrin, Kaarthik Sundar, and Victor M Zavala. Graph-Based Modeling and Decomposition of Energy Infrastructures. *arXiv preprint arXiv:2010.02404*, 2020.

[54] Henrique Silvério, Sebastián Grijalva, Constantin Dalyac, Lucas Leclerc, Peter J Karalekas, Nathan Shammah, Mourad Beji, Louis-Paul Henry, and Loïc Henriet. Pulser: An open-source package for the design of pulse sequences in programmable neutral-atom arrays. *Quantum*, 6:629, 2022.

[55] Hardik Singh, Raavi Sai Venkat, Sweta Swagatika, and Sanjay Saxena. GPU and CUDA in hard computing approaches: analytical review. *Proceedings of ICRIC 2019*, pages 177–196, 2020.

[56] Spencer T Stober, Stuart M Harwood, Dimitar Trenev, Panagiotis Kl Barkoutsos, Tanvi P Gujarati, and Sarah Mostame. Considerations for evaluating thermodynamic properties with hybrid quantum-classical computing work flows. *Physical Review A*, 105(1):012425, 2022.

[57] John E Stone, David J Hardy, Ivan S Ufimtsev, and Klaus Schulten. GPU-accelerated molecular modeling coming of age. *Journal of Molecular Graphics and Modelling*, 29(2):116–125, 2010.

[58] Herb Sutter et al. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's journal*, 30(3):202–210, 2005.

[59] Botond Szilágyi and Zoltán K Nagy. Graphical processing unit (GPU) acceleration for numerical solution of population balance models using high resolution finite volume algorithm. *Computers & Chemical Engineering*, 91:167–181, 2016.

[60] Arthur H Veen. Dataflow machine architecture. *ACM Computing Surveys (CSUR)*, 18(4):365–396, 1986.

[61] Xiangwen Wang, Xianghong Lin, and Xiaochao Dang. Supervised learning in spiking neural networks: A review of algorithms and evaluations. *Neural Networks*, 125:258–280, 2020.

[62] Daniel P Word, Jia Kang, Johan Akesson, and Carl D Laird. Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Computational Optimization and Applications*, 59(3):667–688, 2014.

[63] Wenqiang Zhang, Bin Gao, Jianshi Tang, Peng Yao, Shimeng Yu, Meng-Fan Chang, Hoi-Jun Yoo, He Qian, and Huaqiang Wu. Neuro-inspired computing chips. *Nature electronics*, 3(7): 371–382, 2020.

[64] Yan Zhang, Panagiotis Vouzis, and Nikolaos V Sahinidis. GPU simulations for risk assessment in CO2 geologic sequestration. *Computers & Chemical Engineering*, 35(8):1631–1644, 2011.

[65] Yu Zhu, Sean Legg, and Carl D Laird. Optimal design of cryogenic air separation columns under uncertainty. *Computers & Chemical Engineering*, 34(9):1377–1384, 2010.