What Quantum Can Learn from Classical Computer Engineering

ANNE Y. MATSUURA, Intel Corp Hawthorn Farm, Hillsboro, United States
TIMOTHY MATTSON, Intel Corporation, Hillsboro, United States

Quantum computing represents a paradigm shift requiring reconceptualization of algorithms, architectures, and software. Although much is new, there is much that quantum computing can learn from traditional classical computer engineering. In this Special Issue, we focus on examples of quantum research inspired by classical computer engineering, ranging from quantum machine learning and image processing to the extension of the "multi-core" concept to quantum computing.

Hardware/software co-design is difficult when communities have their unique jargon and disjoint problem domains. For parallel programming on classical computers, an approach for communicating application developer needs emerged around identifying key bottlenecks that limited software. It was difficult to guess features of future applications, but experienced developers knew features of a computer system that would limit software performance. For each feature, a small kernel was developed with the idea that if the kernel ran well, a system would be an effective system for application developers. This set of kernels was called the *Parallel Research Kernels*. They have been useful to guide design of parallel computers. We hypothesize that an analogous tool for quantum computing, called *Quantum Research Kernels*, may aid hardware/software co-design of quantum computing systems. In this article, we motivate the Quantum Research Kernels and provide working examples.

CCS Concepts: • **Hardware → Quantum computation;**

Additional Key Words and Phrases: Hardware/software co-design, parallel computing, architecture

ACM Reference Format:
Anne Y. Matsuura and Timothy Mattson. 2025. What Quantum Can Learn from Classical Computer Engineering. *ACM Trans. Quantum Comput.* 6, 1, Article 1 (January 2025), 6 pages. https://doi.org/10.1145/3705007

1 Introduction

Quantum computers have the potential to transform computing, opening up new classes of applications and promising to solve problems that are intractable for classical computers. To reach that potential, however, requires great technological innovations, including research advances in fundamental physics, the development of new quantum programming models that "normal" humans can use, and solving a host of issues in both software and hardware concerning building quantum systems.

Author's Contact Information: Anne Y. Matsuura, Intel Corp Hawthorn Farm, Hillsboro, Oregon, United States; e-mail: anne.y.matsuura@intel.com; Timothy Mattson, Intel Corporation, Hillsboro, Oregon, United States; e-mail: tim@timmattson.com.

ACM 2643-6817/2025/01-ART1
https://doi.org/10.1145/3705007

Quantum computing has a long way to go, but we believe by leveraging best practices from classical digital computing, quantum computing can advance through the stages of development at a faster rate. This Special Issue focuses on research in the quantum space that has been heavily inspired by classical computer engineering concepts. The articles are nominally divided into two types: (1) manuscripts that focus on optimization of quantum control and quantum architecture using classical design techniques and (2) manuscripts focused on application-driven design methods. In this article, we briefly foreshadow the research contained in this Special Issue, as well as introduce a new concept of our own called **Quantum Research Kernels (QRK)**, analogous to **Parallel Research Kernels (PRK)**, which have proven useful in designing classical parallel computing systems [4].

2 Using Classical Design Techniques for Optimization of Quantum Architectural Design

As noted in Section 1, classical design techniques can offer fruitful directions for optimization of quantum architectural design. For instance, in their article "Switching Time Optimization for Binary Quantum Optimal Control," Fei et al. study a binary quantum control optimization problem and use optimization models and algorithms from classical design to develop a new algorithmic framework that not only optimizes control functions but also controller time-switching points. They show numerically that their approach results in higher quality control with a similar number of switches in significantly less time than using a more standard fixed-time discretization control model.

Similarly, Dobbs et al. also use classical design techniques but in their case to optimize quantum circuit design. In their work "Efficient Quantum Circuit Design with a Standard Cell Approach, with an Application to Neutral Atom Quantum Computers," they use a standard cell approach borrowed from classical circuit design to speed up optimization of circuit layouts. Their approach divides the layout into qubit storage (memory and measurement) zones and standard cell processing zones. They show that, when compared to automatic routing methods, their layout-aware routers reduce resources while achieving shallower and faster circuits. Their approach is most natural on neutral atom qubits, a qubit type that has recently shown rapid progress.

In "Revisiting the Mapping of Quantum Circuits: Entering the Multi-Core Era," Escofet et al. use the classical design concept of "multi-core" and perform exploratory research into finding an optimal scalable multi-core quantum computing architectural design. The authors propose the Hungarian Qubit Assignment approach to execute quantum algorithms across cores while minimizing inter-core communication. The authors show that their multi-core mapping algorithm improves execution time and non-local communications, while also shedding light on the future work needed to enable the efficient execution of quantum algorithms on scaled-up multi-core quantum architectures.

3 Quantum Application-Driven Design

Several of the articles take a more application-driven approach to design. For instance, the work from Yan et al. entitled "Lessons from Twenty Years of Quantum Image Processing" gives a retrospective of the field of quantum image processing, as well as addresses potential limitations and how to overcome bottlenecks in this field as we look to the future. The development of a quantum machine learning algorithm to address current issues in the field, such as insufficient labeled data for important domains like medical imaging processing, is addressed in the article by Don et al., "Q-SupCon: Quantum-Enhanced Supervised Contrastive Learning Architecture within the Representative Learning Framework".

"Improving the Exploitability of Simulated Adiabatic Bifurcation through a Flexible and Open-Source Digital Architecture" by Volpe et al. proposes a digital architecture for emulation of quantum computing. This quantum-inspired approach aims particularly at condensed matter physics, Ising-oriented simulations. It targets implementation on classical platforms like field-programmable arrays.

The article by Mohammadisiahroudi et al. entitled "Improvements to Quantum Interior Point Method (IPM) for Linear Optimization" focuses on applying quantum methods to the classical IPM approach to solving optimization problems. Previous quantum IPM approaches required a large number of quantum gates and qubits. In this article, the authors present a more efficient quantum IPM method to solve linear problems.

"Utilizing classical programming principles in the Intel Quantum SDK: implementation of quantum lattice Boltzmann method" by Tejas et al. demonstrates the importance of quantum computing for simulating complex fluid dynamics problems. The authors illustrate the usage of a modular classical program design implementation and focus on generalizing and expanding the quantum circuits to enable more precise control of variables.

Last, the article by Quetschlich et al., "MQT Predictor: Automatic Device Selection with Device-Specific Circuit Compilation for Quantum Computing," proposes a methodology that selects the quantum device and compiler that is best for the particular quantum application. This emphasis on using the workload for determining the most appropriate pairing of hardware and software illustrates the importance of the needs of workloads to help define both the hardware and the software of a quantum computing system.

4 Introducing the QRK

One lesson learned from the era of digital computing is that the best systems emerge from a software/hardware co-design process. All too often, hardware is designed and then "thrown over the fence" to software developers who have to figure out how to make the hardware useful. It is much better to have the software and hardware teams working side by side so that the programming systems are ready as soon as the hardware is available and the hardware includes features specifically needed to make the software both more efficient to run and easier to write.

However, it is challenging to do hardware/software co-design when you do not know what future applications will look like. Hypothesis: Even though we may not know what future applications will look like, we do know the features of a system that will limit these applications. If we can define these "bottlenecks" and build a system that collectively minimizes their impact, we can be confident that the system will be effective for these future applications.

This was the basic principle behind the PRK [5] in the parallel computing field. The PRK are a collection of simple kernels that expose the features of a system that limits parallel performance. They are small, generate their own data, do a computation (so systems cannot "cheat"), and test their results. In essence, they are a way for application programmers to precisely define what they need from a system, to guide hardware developers to build systems that will work well for applications. They have proven useful for designing systems [4] and to explore the suitability of extreme scale programming models [2].

Here, we would like to launch a conversation about using an approach similar to the PRK for quantum computing. We call these the *QRK*. Can we define features of quantum computing systems that will limit applications for these systems? Can we produce well-defined kernels to expose these features? Can we anticipate the breadth of application design patterns so we can be confident that the set of QRK are complete? These are challenging questions. In this article, we define the problem and propose a process to answer these questions.

Table 1. PRK: A Set of Basic Kernels Designed to Stress Features of a System That Limit the Performance of HPC Applications

Name	Definition	Exposed System Feature
Transpose	Transpose a dense matrix	Bisection bandwidth
Reduce	Elementwise sum of multiple private vectors	Message passing, local memory bandwidth
Sparse	Sparse-matrix vector product	Scatter/gather operations
Random	Random update to a table	Bandwidth to memory with random updates
Synch_global	Global synchronization	Collective synchronization
Synch_p2p	Point-to-point synchronization	Message passing latency, remote atomics
Stencil	Stencil method	Nearest neighbor and asynchronous comm.
Refcount	Update shared or private counters	Mutual exclusion locks
Nstream	Daxpy over large vectors	Peak memory bandwidth
DGEMM	Dense matrix product	Peak floating point perf.
Branch	Inner loop with branches	Misses to the instruction cache, branch prediction
PIC	Particle in cell	Unstructured asynch. multitasking
AMR	Adaptive mesh refinement	Hierarchical asynch. multitasking

4.1 Parallel Research Kernels

The PRK stress a system in ways parallel applications in **High-Performance Computing (HPC)** would. They are listed in Table 1, where we provide the name of each kernel, a brief definition, and the features of a parallel system stressed by the kernel. The kernels are defined mathematically and with a reference implementation using C, OpenMP, and MPI. They are available in a GitHub repository [3].

We submit that the PRK are complete. If a system is built that does well with all of them, then it is likely that the system will be effective for running parallel HPC applications. The PRK were selected by an ad hoc committee of parallel application programmers. When we started the project (in 2005), parallel computing in various forms had been around for more than 25 years. We were able to convene a committee of experts with decades of experience. Over the course of several meetings and long e-mail chains, the committee came up with the list of kernels.

4.2 Quantum Research Kernels

The QRK, inspired by the PRK, will define a set of kernels designed to expose features of a quantum computing system that will constrain the success of applications written for quantum computers. The QRK are intended to be a sufficiently complete set such that if a system were constructed that did well for each of the QRK, then that system would most likely be a successful system for supporting key applications.

As with the PRK, the QRK will be produced through a community-driven process by programmers interested in writing applications for quantum computers. To help nurture this conversation,

we provide an example of a potential QRK. One of the bottlenecks for scalable quantum computing is the difficulty of loading the data (particularly classical data) into the quantum machine. State preparation can be an exponentially hard problem itself, leading to the conundrum that loading of the data into the quantum machine can completely negate the speedup gained by doing the problem on a quantum computer [1].

Hence, the first QRK follows:

— ***Name***: Encode.
— **Definition**: Create a sequence of classical values from $i = 0$ to $i = N$ equal to $4i\pi/N$.
— **Action**: Encode qubits with the values from the previous step. Rotate each qubit by $\pi/6$.
— ***Test***: Read qubits and confirm that they have the correct rotated value.

Note that this has all the features we would expect in a QRK. It defines problem input that is generated so the QRK can scale to any number of qubits. A specific operation is defined so that at the end of the QRK, the result can be validated. Finally, it exposes a specific feature of a quantum computer that will limit the ability of applications to run on the system expressed in terms that architects can understand and use to guide the design of a quantum computer.

We have the beginnings of two additional QRK. The first is one we call the *Computational Area*. This is the product of a number of qubits that can be entangled times the number of operations that can be carried out before the entangled state can no longer be maintained. A high Computational Area can be produced by a small number of qubits that remain entangled over a large number of operations or by having a large number of qubits that remain entangled for a small number of operations. There are advantages to both cases so this measure supports both.

The second additional QRK is called *Parallel Streams*. This measures the ability of a quantum computer to execute multiple independent streams of operations at the same time and in parallel. There are a number of options on how to define the work that must be carried out in parallel. Initially, we would run the Computational Area QRK in each stream, but for a wider range of applications for quantum computers, we may find a better case for each stream.

These three QRK are just a start. We need a larger set that covers the full range of features needed from a successful quantum computer, hopefully on the order of 10. The system features that each QRK stresses overlap, but taken together they need to cover the full range of features needed by application programmers from a quantum computer so that those designing parallel systems can be confident that a system that does well on the full set of QRK will meet the needs of applications programmers. This is the primary goal of the QRK. However, once established, an application designer can model the needs of an application in terms of a linear combination of QRK, thereby using them to help select the quantum computer best suited to a particular application.

5 Conclusion

We believe that the quantum computing field can learn from classical computing design techniques. This Special Issue focuses on a number of examples illustrating this. In addition, we introduced the QRK as a design tool for hardware/software co-design for quantum computing, a similar approach to the successful PRK used in classical computing, and we provided a few examples of QRK. We further call to the quantum computing user community to work together to develop a complete set of QRK that can guide the design and development of quantum computing.

References

[1] Scott Aronson. 2015. Read the fine print. *Nature Physics* 11 (2015), 291–293.
[2] R. F. Van der Wijngaart, A. Kaya, J. R. Hammond, G. Jost, T. St. John, S. Sridharan, T. G. Mattson, J. Abercrombie, and J. Nelson. 2016. Comparing runtime systems with exascale ambitions using the parallel research kernels. In *Proceedings of the International Supercomputing Conference*.

[3] Jeff Hammond. n.d. Parallel Research Kernels (PRK). Retrieved November 20, 2024 from https://github.com/ParRes/Kernels

[4] T. G. Mattson, R. van der Wijngaart, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe. 2010. The 48-core SCC processor: The programmer's view. In *Proceedings of the ACM/IEEE Conference on Supercomputing*.

[5] Rob F. Van der Wijngaart and Timothy G. Mattson. 2014. The Parallel Research Kernels. In *Proceedings of the 2014 IEEE High Performance Extreme Computing Conference (HPEC'14)*. IEEE, 1–6.

Received 7 November 2024; revised 7 November 2024; accepted 11 November 2024