Co-design of Hardware and Algorithms for Real-time Optimization

Eric C. Kerrigan

Abstract— **It is difficult or impossible to separate the performance of an optimization solver from the architecture of the computing system on which the algorithm is implemented. This is particularly true if measurements from a physical system are used to update and solve a sequence of mathematical optimization problems in real-time, such as in control, automation, signal processing and machine learning. In these real-time optimization applications the designer has to trade off computing time, space and energy against each other, while satisfying constraints on the performance and robustness of the resulting cyber-physical system. This paper is an informal introduction to the issues involved when designing the computing hardware and a real-time optimization algorithm at the same time, which can result in systems with efficiencies and performances that are unachievable when designing the sub-systems independently. The co-design process can, in principle, be formulated as a sequence of uncertain and non-smooth optimization problems. In other words, optimizers might be used to design optimizers. Before this can become a reality, new systems theory and numerical methods will have to be developed to solve these co-design problems effectively and reliably.**

I. Do You Want Some Gold?

Imagine I were to give you 50 g of gold and a list of unsorted numbers. You can keep some of the gold by using a pen and paper to add up these numbers, but you have to return 10 g to me for every minute that you do not have the answer. It takes me just under ten minutes to complete this exercise, so you are unlikely to keep any of the gold if you did this on your own. The good news is that your friends can help you, but you have to share the prize amongst yourselves.

Suppose that you are as fast as me at adding up numbers and that you will share the prize equally. If you were to get only one friend to help, then you might be able to complete the task in under five minutes and take home 5 g each, after having returned 40 g to me. If you were to get nine friends to help, then all ten of you might be able to complete the task in just under one minute, each still only taking home 5 g of gold from the pot of 50 g.

There is an optimal number of friends if you want to maximize your individual winnings. If only four friends help, then the five of you might be able to finish in just under two minutes and take home 8 g each from the remaining 40 g.

Would you modify your strategy if you knew that all the numbers were in a small range, e.g. from 1 to 5? Is it easier to count the different types of numbers than adding up each in turn? What would you do if the numbers were sorted?

Think of the strategy in which you split tasks amongst friends as the algorithm and your team members as the computing hardware. The amount of gold that each of us holds are the states of a dynamical system that change with time and is a function of the size and strategy of your team.

If you understand this example, then you appreciate why the performance of an algorithm is a function of the computing hardware. You are likely to agree that it is important to design the algorithm and hardware at the same time, that there are trade-offs to be made and that hardware-algorithm co-design can be formulated as an optimization problem. You probably also understand that if there is a physical system, which affects computations or vice versa, then it may not be sensible to design the computing system without taking into account how the physical system evolves in time.

This paper is tutorial in nature and introduces the issues that a co-designer has to face when using mathematical optimization to solve control, signal processing, automation and machine learning problems in real-time. The development is relatively informal, with a small amount of mathematical detail to partially cater for the technician, and is mostly a collection of problems, ideas and food for thought. There are no references to books or papers, because it is easy to find introductory texts to most of the topics mentioned below and the aim is not to provide a review of solutions. This is instead an attempt at stimulating discussion and research. Co-design for real-time optimization is still young and many fundamental problems remain to be defined.

II. Difficult Decisions

Time does not stop and data is always uncertain. Scientists, engineers, economists, sociologists, business managers and policymakers are regularly faced with having to decide what, where, when and how accurate to measure, store, compute and communicate.

The full potential of computing systems to solve real-world problems can be realized only by co-design, where the algorithm is designed at the same time as the computing hardware, while also assessing the impact this design has on the application for which the computation was intended. This is particularly true when control, signal processing, automation and machine learning algorithms are to be deployed on embedded, distributed and safety-critical systems.

An open question is how best to co-design for real-time optimization applications, in which a sequence of mathematical optimization problems is updated with data obtained from a physical system and decisions are made based on solutions to these optimization problems. The results from these decisions are often also fed back into the physical system, either directly or indirectly, thereby affecting the behavior of the overall cyber-physical system. Real-time

The author is with the Department of Electrical & Electronic Engineering and Department of Aeronautics, Imperial College London, Exhibition Road, London SW7 2AZ, United Kingdom, e.kerrigan@imperial.ac.uk

optimization is used to solve a variety of problems across a huge range of temporal and spatial scales, from nano-positioning devices to power systems and the climate.

There is always a large number of options available for hardware-algorithm co-design. Choices for the computing hardware include:

- Processor: Microcontroller, DSP, GPU, FPGA, PLC or multi-core general-purpose processor?
- Communication: WiFi, Bluetooth, Ethernet or ZigBee?
- Storage: DRAM, SRAM, HDD, SSD or DVD?

A non-exhaustive list of choices for the mathematical optimization algorithm includes the following:

- Interior point or active set?
- Gradient-based or derivative-free?
- First-order or second-order method?
- Exact or inexact solutions to equations?
- Deterministic or stochastic?
- Direct or iterative linear solver?
- Large and sparse or small and dense matrices?

The granularity of a decision could also have a significant effect on overall performance. For example, should the parallelization be at the bit-level or higher and how much error can be tolerated within each loop of the algorithm?

III. Computing Resources are Finite

Computing systems are composed of processing, storage and communication sub-systems:

- *Processing:* This includes low-level arithmetic units, such as multipliers and adders, to higher-level units, such as microprocessors, servers and data centres.
- *Storage:* Data and computations need to be stored, either permanently or temporarily, in units that include on-chip registers, off-chip memory or external drives.
- *Communication:* The processing units need to communicate with each other and the storage sub-systems via wired or wireless networks. This can range from fast on-chip buses to relatively slow and distributed networks.

It is therefore important to take into account the fact that computing takes time, consumes energy and occupies space:

- *Time:* In real-time applications a physical system continues to evolve while the computation is carried out. The timing of a computation is therefore critical in determining the usefulness of a result. For example, a computation can arrive too soon[1] or too late when controlling the ignition in an internal combustion engine.
- *Energy:* Energy is consumed and heat is dissipated every time a transistor switches state, data is stored or transmitted. In many applications the electrical energy or power required by the computing system is the key driving factor during design, e.g. in energy-harvesting wireless sensor networks and big data centres.
- *Space:* This is the physical space and weight of the computing sub-systems, e.g. amount of semiconductor, cabling and transmitter/receiver size. Faster and more accurate computations require many large processing units, memory and communication interconnections. A big, expensive supercomputer can solve more complicated problems than a small, cheap microcontroller.

[1]Computations that arrive too early may need to be stored in a buffer, therefore complicating the design, increasing cost and wasting energy.

In most applications, the above three physical resources not only have to be traded off against each other, but also the performance, robustness and cost of the overall cyber-physical system. For example, it might be possible to (i) speed up a computation by parallelizing an algorithm at the expense of using more processors, (ii) reduce the energy consumption of the computation by decreasing the power supply voltage and processor clock speed, thereby increasing the latency with a subsequent deterioration in system performance and robustness, or (iii) reduce the silicon area by using arithmetic units with fewer bits, but having to take more iterations of an algorithm in order to guarantee that numerical errors are within acceptable limits.

A co-designer therefore has to understand how best to map an algorithm to processing, storage and communication sub-systems in order to meet constraints on time, space, energy, cost, performance and robustness.

IV. Trade-offs Have to be Made When Adding

Consider, as an example, the problem of computing the sum $s := \sum_{i=1}^{n} x_i$ of the components of a vector of integers $x := (x_1, \ldots, x_n)$. There is clearly a large number of algorithms and computer architectures that could be designed to solve this problem, each requiring different amounts of time, energy and space.

It is easy to produce a design that adds all numbers sequentially, as in Algorithm 1, with only one scalar adder. Using this algorithm, it follows that the result will be available after $n-1$ iterations of the while loop.

Algorithm 1 Sequential addition

```
s ← x_1
i ← 2
while i ≤ n do
    s ← s + x_i
    i ← i + 1
end while
```

Suppose now that p scalar adder units are available. Based on Algorithm 2, a computing system can be designed that significantly reduces the time taken to compute s by adding up to p pairs of scalars in parallel during each iteration of the while loop. It is possible to show that the result will be stored in the variable s after only $\lceil \log_2 n \rceil$ iterations if $p \geq \lfloor n/2 \rfloor$ and after

$$r(n,p) := \lfloor n/p \rfloor - 1 + \lceil \log_2(p + n \bmod p) \rceil$$

iterations if $p < \lfloor n/2 \rfloor$, where $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote the ceiling and floor operators, respectively. If there is only one scalar adder, i.e. $p = 1$, then $r(n,1) = n-1$ iterations of the while loop will be executed, as with Algorithm 1.

Algorithm 2 Parallel addition

$s \leftarrow x$
$i \leftarrow n$
while $i > 1$ **do**
  $\ell \leftarrow \min\{\lfloor i/2 \rfloor, p\}$
  $q \leftarrow (s_1, \ldots, s_\ell) + (s_{\ell+1}, \ldots, s_{2\ell})$ using ℓ adders
  $s \leftarrow (q, s_{2\ell+1}, \ldots, s_i)$
  $i \leftarrow i - \ell$
end while

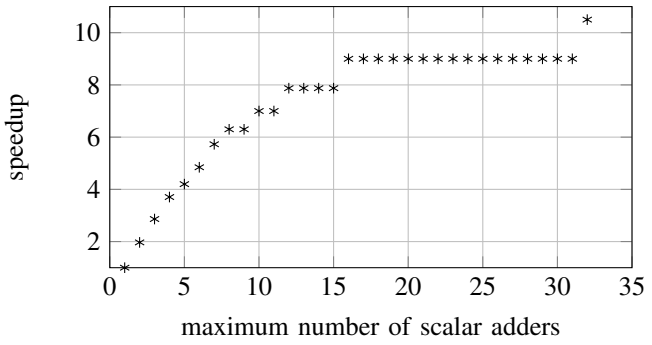

Fig. 1. Speedup of parallel addition as a function of the maximum number of scalar adders p when adding $n = 64$ numbers.

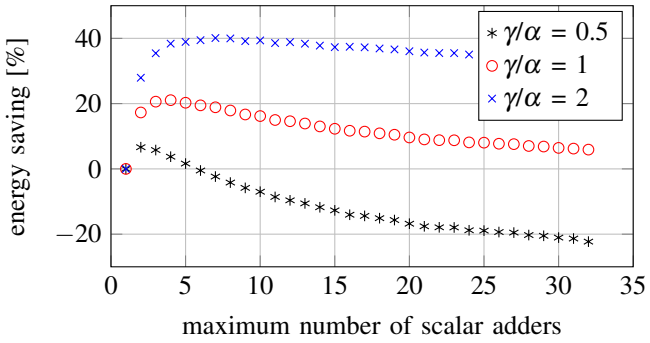

Fig. 2. Energy saved for parallel addition as a function of the maximum number of scalar adders p for different values of γ/α if $n = 64$ and $\beta = 1.2$

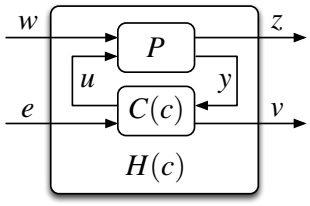

Fig. 3. Cyber-physical system $H(c)$ with the inputs and outputs of the physical sub-system P and computing sub-system $C(c)$.

It is not obvious from Algorithm 1 whether it is possible to make efficient use of parallel architectures, whereas this is much clearer in Algorithm 2. It is therefore essential to consider how best to describe an algorithm in order to allow for efficient use of computing resources and to be able to compute the fundamental limits of performance.

Suppose we wish to compute the speedup $r(n,1)/r(n,p)$ in completing all iterations of the while loop in Algorithm 2 as the number of adders increases. For simplicity, assume that each iteration takes the same amount of time. Figure 1 shows that the speedup, as a function of p, is less than linear due to diminishing returns with increasing p. The figure shows that the maximum speedup for $n = 64$ is attained when $p = 32$ and $r(n,p) = \lceil \log_2 n \rceil = 6$. Note also that the speedup is constant over a relatively large interval, i.e. $16 \leq p \leq 31$. This kind of analysis is therefore very useful to predict how best to trade off space against time so that resources are not wasted.

A very important observation to make is that the energy consumed during the computation can be decreased by increasing the number of adders. This is because the time taken to compute the solution can decrease sufficiently fast relative to the additional energy consumed per iteration. Energy efficiency is very important in many applications, ranging from small, embedded computing systems to large, distributed super-computing systems.

Suppose that the energy consumed by the while loop in Algorithm 2 can be approximated by a function of the form

$$E(n,p) := \sum_{k=1}^{r(n,p)} \left(\alpha \ell_k^\beta + \gamma \right),$$

where ℓ_k is the number of scalar adders used at the k^{th} iteration. The constant $\gamma > 0$ is the energy consumed by all arithmetic and logical units that do not scale with the number of adders. The term $\alpha \ell_k^\beta$ represents the energy consumed by a single scalar addition during the k^{th} iteration, where the constant $\alpha > 0$ is a function of the geometry and material used to construct the adders. The exponent β is larger than 1 in practice, because the time and space taken by some of the computations and circuitry scale nonlinearly with ℓ_k.

Figure 2 is a plot of the percentage energy saved $100[1 - E(n,p)/E(n,1)]$ as a function of p for different values of γ/α if $n = 64$ and $\beta = 1.2$. The energy consumption first improves as p increases and reaches an optimal value for some value of p before gradually worsening. Note also that the relative energy saving improves as γ/α increases. Figures 1 and 2 clearly demonstrate that there is a trade-off between time, space and energy when adding up a set of numbers.

For many problems it is important to also include the communication network and memory architecture in the space-time-energy analysis. There is considerable scope here for creative solutions and this is therefore left as an exercise.

V. Optimizing in Real-Time with Uncertain Data

Real-time optimization is challenging, because it involves solving a sequence of optimization problems that are functions of the measurements and dynamics of a physical system, both of which are uncertain. The computing system, which solves the sequence of optimization problems, is also an uncertain dynamical system. The physical system affects the computations and/or vice versa, therefore it is important to consider the behavior of the overall cyber-physical system when designing the algorithm and computing architecture.

Suppose that, as in Figure 3, we have a physical, causal

dynamical system

$$P : (u, w) \mapsto (y, z)$$

with measured output $y : \mathbb{R} \to \mathscr{Y}$ and manipulated/control input $u : \mathbb{R} \to \mathscr{U}$. The output $z : \mathbb{R} \to \mathscr{Z}$ includes variables that we wish to estimate, optimize, regulate or constrain. The input $w : \mathbb{R} \to \mathscr{W}$ represents unknowns, which include measurement noise, disturbances, time-varying references, parameter uncertainties and unmodelled dynamics.

In real-time optimization, the problems that need to be solved at various time instants are typically in the form

$$\theta^*(d) \in \arg\min_{\theta} \{V(\theta, d) \mid (\theta, d) \in A\}, \tag{1}$$

where $\theta^*(d)$ is a vector or function to be computed, V is a given cost function, A is a given set and d is some given data. The data d includes current or past measurements and the solution $\theta^*(d)$ can include estimates of z. In many applications, part of the solution $\theta^*(d)$ is fed back to the physical system in order to update u. The data d also often includes the current time, results from previous computations, a model of the physical system and an uncertainty model.

The result returned by the computing system is often inaccurate, e.g. due to finite precision arithmetic errors, early termination of the algorithm or because the optimization problem is non-convex and the solver could only compute a locally optimal point. It follows that the computing system, which contains the optimization solver, is a strictly causal dynamical system

$$C(c) : (y, e) \mapsto (u, v),$$

where the input $e : \mathbb{R} \to \mathscr{E}$ represents computational errors and the output $v : \mathbb{R} \to \mathscr{V}$ contains results of computations, including estimates of the accuracy and precision of these computations. The variable c contains the design parameters and choices that need to be made for the algorithm and hardware, e.g. in Section IV we had $c := p$.

A good designer would evaluate the performance of an algorithm and computing architecture by including the physical system in the analysis. The aim of co-design is therefore to choose c such that the combined cyber-physical system

$$H(c) : (w, e) \mapsto (z, v)$$

satisfies given constraints on performance, robustness, stability and cost.

VI. More Difficult Decisions

There is considerable freedom in the choice of what to include in the design variable c. In addition to the choices listed in Section II, below is a small selection of items that can have a significant impact on the final design:

- Hardware:
  - Allocated cost, space, weight, energy and power.
  - Number of processors, cores or arithmetic units.
  - Memory architecture, including latency and size.
  - Communication architecture, including quantization and bandwidth.
  - Measurement and actuation architecture, including quantization and sampling, e.g. time-triggered (period or aperiodic) or event-triggered.
  - Pipeline depth for trading off throughput against latency, energy and space.
  - Number representation, e.g. fixed point or floating point, including number of bits.
  - Processor clock frequency and supply voltage.
- Algorithm:
  - Precision, accuracy and termination tolerance, e.g. when computing gradients, solving linear equations or evaluating the sub-optimality of an iterate.
  - Number of iterations for each loop.
  - Parameters for computing step length.
  - Scaling of data and preconditioning of matrices.
  - Amount of measurements to store.
  - Amount of computations to store.
  - Complexity of the physical model, e.g. model order, sparsity, delays, nonlinearities, coordinate system.
  - Length of past and future time horizons for estimating or predicting the physical system response.
  - Coarseness of the discretization when approximating solutions to differential equations.
  - Strategy for scheduling computing tasks and resources, including communication protocol.

Co-design is complicated by the fact that, though some of the items listed above are continuous variables, e.g. sample period or time horizon, many design variables are discrete.

VII. Uncertainty as Both Enemy and Friend

In the examples above we have assumed that no numerical errors were introduced during each addition and that the data was accurate. As mentioned in Section V, it is always the case in practice that the data is uncertain and errors are introduced during computations.

If data is uncertain then it is possible to destabilize or degrade the system performance if care is not taken during the computation. The natural response for most is therefore to treat uncertainty as a problem, rather than seeing this as an opportunity to design efficient systems, as discussed below.

The accuracy, precision and timeliness of a computation can be traded off against each other — an approximate answer today might be preferable over an accurate answer tomorrow, when there may no longer be any gold left. When is it necessary to compute with 16 significant digits when the data is only measured to 3 significant digits?

The problem therefore becomes really interesting if one has to decide on the number representation and sequence of operations in order to provide guarantees on accuracy and precision. Prior information on the data can also be used to develop deterministic and stochastic algorithms that can be terminated at any time with an estimate of the result and associated error, which reduces upon further iteration.

A. Mistakes are Allowed

Consider again the problem of winning some gold by adding up a list of numbers and suppose now that an incorrect

answer is acceptable. Upon receiving an answer I will work out the percentage error and ask you to return to me the same percentage of gold left in the pot. It is up to you to decide how much you want to trade off accuracy against time and the amount of gold your team takes home.

Would you add up small numbers first or start with the large numbers? What would you do if the numbers were correct to only two significant digits? I cannot expect you to give me an accurate answer, but would accept an estimate with bounds on the error.

Clearly, it is possible to add uncertain numbers in a structured list in significantly less time, compared to when there is no uncertainty, the list has no structure and errors are unacceptable.

B. Exploiting Uncertainty in Real-time Optimization

The above example suggests that uncertainty can be exploited when wishing to design an efficient computing system. However, it is not always clear how best to do this for real-time optimization.

The most important observation to make for real-time optimization is that there is feedback. Measurements y of the physical system are used to correct for errors by updating the data d of the sequence of optimization problems. In control and automation applications the solution $\theta^*(d)$ is used to update the manipulated/control input u. In signal processing and machine learning applications the aim is to produce estimates of variables for which measurements are unavailable or noisy — this can be modeled by u containing the estimates and z the errors in the estimates, which are minimized or constrained via real-time optimization.

The control theory literature contains many fundamental concepts, such as the gap metric, to understand what impact uncertainty and feedback has on a system. Control theory can be used to exploit and justify the fact that a model for designing a feedback algorithm is not necessarily a good model for open-loop simulation and vice versa. For example, linearized and reduced-order models can be used to decrease the computational effort required, while improving system performance and robustness, even if the actual physical system is nonlinear and infinite dimensional.

Experts on sampled-data control and digital signal processing appreciate that accuracy and time can be traded off against performance and robustness. Judged from the perspective of the interconnected cyber-physical system, it might be better to use computationally efficient algorithms with feedback at a high rate, compared to using advanced algorithms with feedback at a slow rate. If the computation takes too long, disturbances might drive the system too far from the prediction for the computed result to be useful.

In real-time optimization, the explicit use of feedback principles to co-design efficient algorithms and hardware is not always exploited to its fullest potential. In many cases the performance of an algorithm or efficiency of the hardware implementation are treated as separate problems without referring back to the intended use of the computation. The performance of the computing system should instead be assessed on the behavior of the interconnected cyber-physical system. Though there has been some research in this area, there is considerable scope for fundamental contributions, especially with regards co-design.

Feedback is perhaps the only systematic method for dealing with uncertainty. Real-time optimization without feedback is not real-time optimization.

C. Known Unknowns

Feedback should not be used if it is not clear what the uncertainties are. Feedback should not be used if the only reason is to reduce computational effort. Feedback can destabilize an open-loop stable system or inject noise where there is none. Real-time optimization should therefore be used only if it is being used to address uncertainties and ideally only if there is a supporting theory to understand how feedback affects performance and robustness.

Feedback theory for linear dynamical systems with linear algorithms is reasonably mature, but this is not the case for real-time optimization algorithms, which are nonlinear and iterative. Though great strides have recently been made to develop such a theory, much remains to be done.

The literature on robust control and robust optimization, where both deterministic and stochastic uncertainty models and algorithms have been developed, contain many ideas that are relevant to real-time optimization. The lessons to be learnt from these fields are arguably that:

1) there is always a trade-off between system performance, robustness and computational resources, and
2) models of the physical system and uncertainties are always just that and nothing more.

As Rumsfeld would say: in any robust design there are known knowns, known unknowns and unknown unknowns.

VIII. Co-design as Multi-Objective Optimization

Traditionally, co-design is done in a manual or ad-hoc fashion, where design parameters are 'tuned'. However, there is an increasing amount of research aimed at developing methods that will make co-design automatic and systematic.

Computing resources, such as time, space and energy, have to be traded off not just against each other, but also against the performance and robustness of the combined cyber-physical system, as illustrated in Figure 4. There is seldom just a single cost function that the designer has to optimize. Furthermore, in practice the assessment criteria are most naturally posed as a set of constraints and it is not known at the start whether these can be satisfied.

As with most engineering design tasks, hardware-algorithm co-design can therefore be formulated as a sequence of constrained, multi-objective optimization problems, where each problem can be written in the form

$$\min_{c}\{F(H(c),c) \mid (H(c),c) \in G\}, \tag{2}$$

where F is a *vector* of operators and G is a set that captures constraints on the design parameters c and behavior of the cyber-physical system $H(c)$. This formalism can capture both

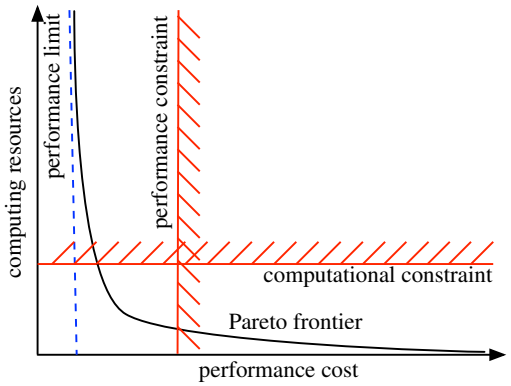

Fig. 4. Illustration of trade-offs in co-design, where the goal is to minimize computational resources and some cost function of the system performance, subject to given constraints. The set of acceptable solutions are on the subset of the Pareto frontier below the computational constraint and to the left of the performance constraint.

deterministic and stochastic assumptions on the uncertainties (w,e), as well as deterministic and stochastic criteria on the outputs (z,v). The question is how best to formulate the sequence of F and G for certain classes of real-time optimization applications and how to solve these co-design problems reliably and effectively.

IX. Solving Co-design Problems

The main technical challenge in formulating co-design problems is to merge abstractions from the physical world with computer science. The study of physical systems is based on differential or difference equations, continuous mathematics and analog data, whereas the study of computing systems is based on logic operations, discrete mathematics and digital data. Unless care is taken, gradient-based optimization solvers might not be suitable for solving co-design problems, due to the interaction between the continuous and discrete dynamics.

Recall the equations for the number of iterations and energy used by Algorithm 2. Because of the floor and ceiling operators, r and E are non-smooth functions, even if n and p are allowed to be real-valued. Many of the functions in hardware-algorithm co-design are non-smooth or can be modeled as smooth functions with noise, hence approximating gradients with finite differences can be unreliable.

Co-design for real-time optimization is further complicated by the fact that the solution to the optimization problem (1) is, in general, a non-smooth function of the data d. This is true even if the cost function V is linear or quadratic and (θ,d) has to satisfy linear inequality constraints.

Added into the bag of challenges is the fact that there are uncertainties (w,e). The optimization problem (2) is therefore an uncertain, non-smooth, mixed-integer problem. Ouch.

Since the design parameter c contains both continuous and discrete variables, the most natural way to formulate the co-design problem (2) is as a mixed-integer nonlinear programming problem, which can potentially be solved using gradient-based solvers. For some applications it might be possible to model all details, including derivatives for the cost and constraint functions. However, in practice this is not as easy as it sounds.

A design based on analytically-derived cost and constraint functions can only give an approximate answer and neglects many other aspects. In practice, electronic design automation tools are used in conjunction with high fidelity software-in-the-loop and hardware-in-the-loop tests when refining, validating and prototyping the design. These simulations and experiments can be very costly and time-consuming. Furthermore, obtaining analytical expressions for the cost and constraint functions, including their derivatives, is often an impossible task due to limitations in the code or hardware.

There has recently been considerable activity in the fields of derivate-free optimization and optimal experiment design. The research in these areas is partially motivated by challenging multi-disciplinary design optimization problems, where analytical derivatives and function values are difficult, expensive or impossible to obtain, as well as highly uncertain. Derivative-free methods use the function values at a set of points to determine the next point without approximating the gradient. The fields of derivative-free optimization and optimal experiment design might therefore provide effective methods for solving certain classes of hardware-algorithm co-design problems, especially in the later stages.

X. Take-Home Message

Computing resources, such as time, energy and space, have to be traded off against each other in order to design an efficient real-time optimizer. Because time does not stop and there are uncertainties in both the physical and computing system, a given design has to be assessed based on the behavior of the interconnected cyber-physical system. Feedback can be used at every level of the design in order to not only reduce computational resources, but also improve system performance and robustness.

Hardware-algorithm co-design can be formulated as a sequence of constrained multi-objective optimization problems. However, these are very difficult to solve using off-the-shelf gradient-based solvers, due to the the uncertain hybrid dynamics of the cyber-physical system. Mathematical analysis, coupled with mixed-integer and non-smooth optimization solvers, could be a good starting point to compute fundamental limits and narrow down the set of possible designs and degrees of freedom. The design might be refined, validated and prototyped while applying techniques from derivative-free optimization and optimal experiment design.

Much research remains to be done in the co-design arena. Though the degrees of freedom, relative importance of objectives and constraints and details are application-dependent, many of the fundamental principles and techniques to be developed could be made to be generic, if care is taken.

Acknowledgments

I would like to thank George Constantinides, Andrea Suardi and Juan Jerez for many interesting discussions on digital computation and co-design.