Interconnect: A Fundamental Constraint

André DeHon

Our ability to build machines that automatically process information is intimately tied to the properties of our physical world. Consequently, the fundamental properties and limits of our physical media will often define the nature of the design space in which we build computing devices. This shows up most clearly when we need to interconnect physically distinct computational building blocks.

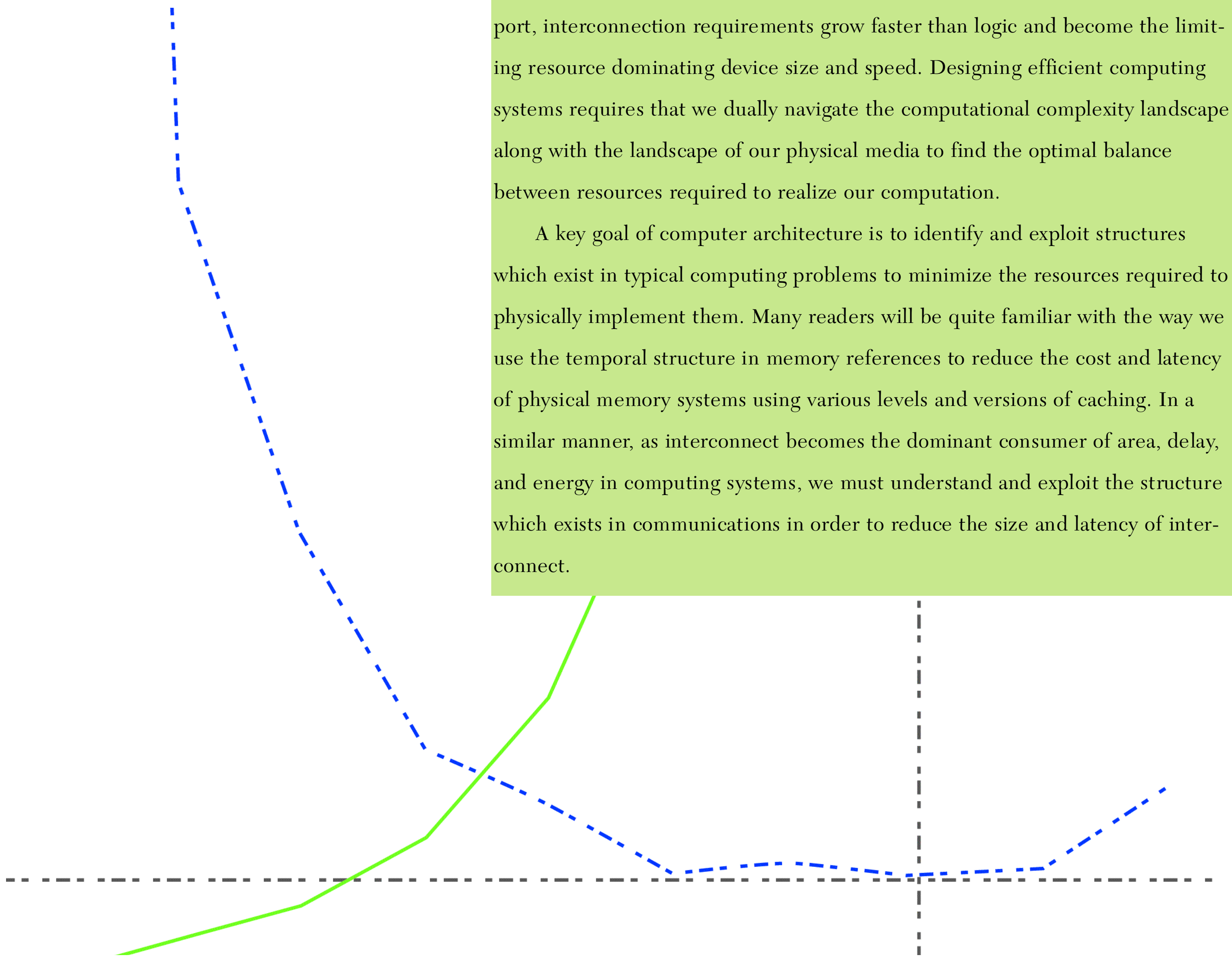
Computing devices are made up of collections of elements (relays, vacuum tubes, transistors, gates, processors, and so on) which must be linked together if they are to cooperate in order to implement some larger computation or computational structure. The *interconnect* which links these elements has become one of the most important aspects of modern computing devices and has opened a rich engineering design space.

A modern example of a universal programmable computing device is the Field-Programmable Gate Array (FPGA). FPGAs are computing devices that use programmable interconnects to allow the end user to wire up a collection of programmable gates (single-output boolean functions of a few inputs, such as NAND, AND, or OR) in an almost arbitrary pattern. They provide a direct way to programmably implement a logical netlist of boolean gates. In this manner, they allow a user to implement almost any computational function just by programming the gates and interconnect.

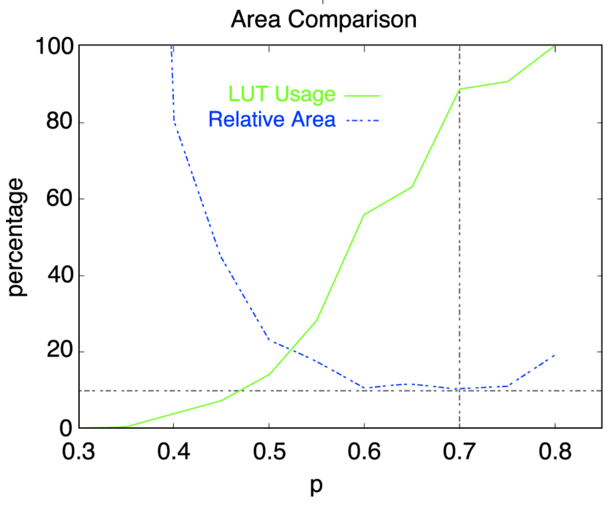
In the design of Field-Programmable Gate Arrays and related spatial computing devices, *interconnect has always been the dominant component* consuming the greatest amounts of area, delay, and energy. With interconnection requirements scaling faster than linearly in device gate capacity and fundamental interconnection delays in VLSI scaling slower than gate speeds, *this problem is only exacerbated as we go forward.* Further, given the size of silicon systems we can build today—and considering the molecular and biological systems we may be able to build in the near future—all kinds of single-component architectures (e.g., multiprocessor, system-on-a-chip, VLIW, Vector, PIM) will be moving toward greater on-chip parallelism and hence greater use of on-chip, programmable interconnect.

The dominance of interconnect arises naturally from the interaction between the properties of our physical world and the structure of our computational tasks. The limits of two- or three-dimensional space for wiring and finite wire widths, combined with the logical communication structure between computing elements, dictates the distances between elements and the space required for interconnect. Since typical, logical communication structures have interconnection demands that are greater than two-dimensional VLSI substrates naturally support, interconnection requirements grow faster than logic and become the limiting resource dominating device size and speed. Designing efficient computing systems requires that we dually navigate the computational complexity landscape along with the landscape of our physical media to find the optimal balance between resources required to realize our computation.

A key goal of computer architecture is to identify and exploit structures which exist in typical computing problems to minimize the resources required to physically implement them. Many readers will be quite familiar with the way we use the temporal structure in memory references to reduce the cost and latency of physical memory systems using various levels and versions of caching. In a similar manner, as interconnect becomes the dominant consumer of area, delay, and energy in computing systems, we must understand and exploit the structure which exists in communications in order to reduce the size and latency of interconnect.



Here, we show how area correlates with LUT (gate) utilization for a *single design* as we increase the interconnect richness (quantified here by the Rent parameter, p). Initially, as we increase richness the area decreases. However, after a point ($p = 0.7$ shown here), increased interconnect richness leads to increased area. What is significant to note in this graph is that the point of minimum area does *not* correspond to the point at which we reach 100% gate utilization. Rather, we achieve minimum area when the gate utilization is only 85%. We actually get a smaller implementation by allowing ourselves to "waste" some gates in order to use the dominant resource (interconnect) more efficiently. In this case, the 100% gate utilization point requires *twice the area* of the depopulated, minimum area point.



Our work has already demonstrated that one key implication of the dominance of interconnect is that, counter to popular intuition, we often want to design our programmable computing structure with a modest amount of interconnect and leave processing elements unused when the interconnection requirements of the task exceed that provided by the substrate. Quite simply, we realize that we have, at least, two distinct commodities that we must balance in our designs: interconnect and computing blocks. Since the balance of these resources is not the same from task to task, or even within a task, it is most efficient to design the substrate to optimize the dominant component, in this case interconnect, at the possible expense of wasting some of the non-dominant resources, in this case computing blocks. Note that this is counter to conventional approaches which seek to provide sufficient interconnection resources such that interconnection limitations can be ignored. The conventional approach is particularly troublesome since interconnect requirements can, potentially, grow as the square of the computing requirements in two-dimensional space. Almost all practical computers in use today rely on this conventional approach.

It is, consequently, imperative that we understand and exploit the structure inherent in our tasks and the structure of our physical media to make any substantial advances in this realm, using the minimum resources necessary for a given application. In this vein, we are working to understand fundamental requirements for both wiring and switching area in limited-dimensional space (e.g., two-dimensional VLSI), including latency–area trade-offs and capacity–routablity trade-offs. We are also working to understand how communication graph topology forces components to be far apart and hence increases communication latency. As we add metal layers to our VLSI processes, but force active components to remain on a single substrate plane, we need to understand how multiple metalization layers change our landscape and how to best exploit this structure.

We are further working to understand how our physical landscape changes as we move to molecular-scale building blocks. It is already clear that molecular-scale devices may have a radically different computing-resource cost structure than conventional VLSI. We have seen switches which can be placed in the space of a wire crossing, whereas conventional VLSI switches require 50–100 times the area of a single wire crossing. Wire resistance may become virtually independent of wire length with ballistic electron transport in molecular wires. And, these structures may eventually allow us to build truly three-dimensional circuits with active elements not limited to a single, two-dimensional substrate plane. These differences may help us surpass some of the interconnection limitations in conventional devices—and they will certainly require new architectures to fully exploit the distinct computing-resource cost structure of these physical substrates.

The automated processing of information continues to radically transform the way we do science and engineering, and the way we live our lives. The limits on our ability to pack computation into small space, with limited energy and maximal speed, sets a fundamental limit on how we can control the world, what level of control we have, and in fact what we can create. Hence, understanding the deep relation between physical computing media and computational tasks remains essential.

A*ndré DeHon is Assistant Professor of Computer Science with expertise in physical implementation of computation, including physical substrates (VLSI, molecular, etc.), programmable media (FPGAs, processors), mapping (compilation, CAD), system abstractions and dynamic management (run-time systems, operating systems, scheduling), and problem capture (programming languages).*

Visit André DeHon's research page at
http://www.cs.caltech.edu/~andre