Physical Algorithms

Roger Wattenhofer

Computer Engineering and Networks Laboratory TIK
ETH Zurich, 8092 Zurich, Switzerland
{wattenhofer}@tik.ee.ethz.ch

Abstract. This is the accompanying paper to an ICALP 2010 invited talk, intending to encourage research in *physical algorithms*. The area of physical algorithms deals with *networked systems of active agents*. These agents have access to limited information for varying reasons; examples are communication constraints, evolving topologies, various types of faults and dynamics. The networked systems we envision include traditional computer networks, but also more generally networked systems, such as social networks, highly dynamic and mobile networks, or even networks of entities such as cars or ants. In other words, the world is becoming algorithmic, and we need the means to analyze this world!

1 Introduction

Computer science is about to undergo major changes because of the ongoing multi-core revolution.[1] These changes will happen on several levels, quite obviously with respect to hardware, but probably multi-cores will affect software and applications as well. We believe that this is a moment of opportunity to do some soul searching, and to reconsider the foundations of computer science. In particular, we suggest to widen the base of computer science, towards parallelism and distributed systems, and as we will explain below, more generally towards physical sciences.

1.1 Algorithms

Studying and analyzing algorithms has been a research success story. Turing machines, along with other machines models, gave way to analyze the efficiency of computing problems, eventually resulting in a beautiful theory with long standing open problems such as "P vs. NP".

The key to this success was essentially abstraction. Even though there is no generally accepted definition of the term "algorithm", when it comes to the analysis of algorithms there clearly is a mainstream, namely that an algorithm

[1] Looking at microprocessor clock speed charts, one may conclude that traditional sequential algorithms had an expiration date around 2005. Since 2005 computers are mostly getting faster because of multiple cores, and hence increased parallelism.

is a recipe that computes an output as a function of an input. Usually, one is interested in minimizing the number of machine operations.[2]

Algorithm analysis is somewhere between engineering and mathematics. Indeed, algorithm analysis seems to have added an extra dimension to mathematics – *efficiency*. Until recently, mathematics mostly cared whether or not a problem had a solution. Now the efficiency dimension adds shades of grey, as one may wonder how difficult it is to find a solution of a problem.

Computational complexity has been a veritable success story in computer science, and subsequently in mathematics, too. The practical impact of the theoretical results is a dense net of problems whose complexity in terms of the input size is known, thus permitting to estimate the required amount of resources to solve a given task. In the end, it allows to decide whether it is *economically* feasible (rather than just theoretically possible) to realize a certain solution to a problem. The success of this approach primarily comes from the simplicity and generality of the underlying input/output computational model. As such, results are comparable and robust against model changes.

In summary, computational complexity is a strong tool to analyze input/output algorithms, applied by computer science as well as other science disciplines. Not only is it a beautiful theory, it also has major practical relevance as computers are by and large equivalent to the theoretical machine models such as random access or Turing machines. Alas, ...

1.2 The King is Dead

... as we speak, computers are changing! Physical constraints prevent sequential systems from getting faster. The speed of light limits processor *clock speeds* depending on the size of the CPU, while in turn transistors cannot be miniaturized arbitrarily due to quantum effects. Instead, hardware designers have turned to *multi-core* architectures, in which multiple processing cores are included on each chip. Today, two or four cores are standard, but soon enough the standard will be eight and more. This switch to multi-core architectures promises increased parallelism, but not increased single-thread performance. Software developers are expected to handle this increased parallelism, i.e., they are expected to write software that exploits the multi-core architecture by consisting of several components that can run in parallel without introducing substantial overhead. This is a notoriously difficult job.

Consequently, it is questionable whether the success story of sequential complexity analysis will continue. Multi-core machines are not the same as random access machines, with an exponentially growing number of cores less and less so. More generally, computing systems become more and more distributed in the sense that information is *local*. Be it the Internet or a system-on-a-chip, no

[2] There are other measures of complexity, e.g. how much space or randomness is used, however, step complexity is generally considered the most important measure. Also, typically only asymptotics are considered, i.e., how much longer the computation takes if the input gets larger.

single part of the system has access to the global state as a whole, fundamentally changing what can—or rather cannot—be done.

Essentially, the question we want to raise is whether the beauty of computational complexity continues to reflect the complexity of our physical world, or whether it will be diminished to a concept of pure theory and math.

1.3 Long Live The King

We claim that algorithm analysis needs to be adapted, to meet the requirements of current and future computer systems. Indeed, we believe that the time is right to ask this question in a more general context. It seems that more and more other sciences are dealing with systems that are distributed in one way or another. In this paper, we survey a few examples; in lack of a better name we call the area *physical algorithms*. The idea is to take techniques from algorithm analysis and computation complexity, and transform these into tools applicable also in settings which do not match the original input/output model.

2 Physical Algorithms

The area of *physical algorithms* deals with networked systems of active agents. These agents are limited in various ways; examples are communication constraints, computational or memory constraints, evolving topologies, various types of faults and dynamics. The networked systems we envision include traditional computer networks such as the Internet or clusters, but also more generally networked systems, such as social networks, or even highly dynamic and mobile "networks" of entities such as cars or ants. Often, parts of the system can be designed or influenced, but others cannot.

Our definition of physical algorithms includes, but is not limited to, the area of distributed algorithms. For instance, we may not be able to specify the protocol of any of the agents at all, instead examining the effect of changing the amount or reliability of information available to the agents. We emphasize dynamics, i.e., algorithms should (if possible) adapt to dynamic changes. In general, we attempt not to presume a potentially unrealistic amount of control of system properties.

Moreover, networks that are large (on their respective scale), suffer from the unreliability of information. Parts of the system may fail, or even behave maliciously in order to corrupt the available data. Information becomes outdated because of changing topology or inputs, or more subtle variations such as differing communication delays. Widening the scope, many systems comprise intelligent agents that act on their own behalf, not necessarily seeking to support the community. Agents may compete for resources selfishly, and cannot be expected to adhere to a specific protocol unless it maximizes their individual profit.

Physical algorithms cover e.g. distributed algorithmic game theory, networks and locality, self-organization, dynamic systems, social networks, control theory,

wireless networks, multi-core systems, and – getting more ambitious – the human brain as a network of neurons [KPSS10], social insects, biological systems in general [Cha09], or also financial and political systems [ABBG10]. Figure 1 provides a "map" of what we consider physical algorithms. The general aim is to find connections between these topics, and to find a general theory that includes central aspects.

In the following, we first give a refined picture of the two axes of Figure 1. We also give a few concrete application areas of physical algorithms, to render the definition more lucid. Some of these areas are well-established already, some are developing rapidly right now.

2.1 Agents

The agents themselves can be limited in different ways. They may be constrained in computational power, in the sense that they can only compute restricted functions, or they may not be able to store (large amounts of) information. Such limitations may be particularly interesting in areas such as social insects.

Fault Tolerance: Moreover, agents may not be reliable. Agents may crash at any point in time, because of lacking energy for instance. Likewise, agents may crash and later regenerate. Also, agents may experience some kind of omission failure, e.g. failing to receive a message, or transient commission failures, e.g. a local state corruption.

Sometimes however, agents will misbehave in more awkward ways, for instance by starting to behave in erratic ways, e.g. by sending random messages. Indeed, the pioneers in fault-tolerance observed machine behavior that was essentially inexplicable. They decided that the only reasonable way to model such behavior was to consider the machines being "malicious". Following their early papers we call such behavior *Byzantine* today [SPL80,LSP82].

When a Byzantine failure has occurred, the system may respond in any unpredictable way, unless it is designed to have Byzantine fault tolerance. In a Byzantine fault tolerant algorithm, agents must take counter-measures that deal with Byzantine behavior. Essentially, in many problems, the fraction of Byzantine agents must be strictly less than a third [LSP82].

More recently, different kinds of agent (mis)behavior are getting into the spotlight, in particular selfishness.

Game Theory: Algorithmic research has often dealt with models and problems that do not follow the orthodox input/output format. Indeed, these studies are probably as old as computer science itself. One of the very fathers of computer science itself, John von Neumann, is among the pioneers of a conceptually significantly different approach.

Before writing one of the earliest articles on computer science [vN93] in 1945, von Neumann (together with Oscar Morgenstern) published *Games and Economic Behavior* [vNM47]. Today, algorithms and game theory are converging

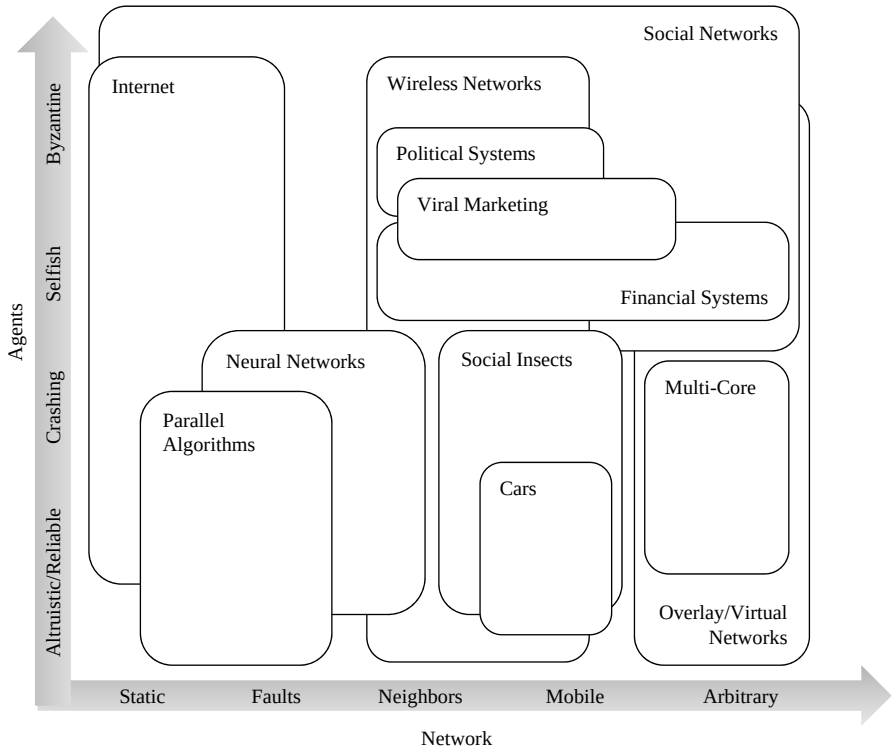

Fig. 1. A representation of the playing field of physical algorithms. On the vertical axis, we see some typical models for the agents that make up networks. In the simplest case, all agents are benevolent and reliable. In many systems, however, agents will be faulty, and for instance crash. In other systems, agents will be selfish in a game-theoretic way, i.e. they will be strategic in order to maximize their benefit. Finally, agents may be malicious (Byzantine), even to a degree that they want to harm the system. Clearly there is no total ordering between these agent models, in some systems faulty agents may be more difficult to deal with than selfish agents. Also, mixed forms of agents may be considered, i.e. some selfish, others malicious. The horizontal axis represents the dynamics of the network. Also here the variants are by no means complete, they should just give some intuition of what is possible. The simplest form of dynamics are no dynamics at all, i.e. fixed networks. In most systems, networks are not static, but allow for at least some slow form of dynamics, for instance, if once in a while—very rarely—a link will fail because of a hardware failure. Further to the right, the frequency of topology changes increases to a point where the network is continuously transforming. The level of dynamics depends on this frequency, but also how these topology changes are restricted. Maybe only local neighborhoods are changing? Maybe the agents themselves are mobile, forming edges whenever two agents are in vicinity of each other? Finally, we may consider completely virtual networks just modeling some physical process, with rather arbitrary forms of dynamics. The figure exemplarily shows typical application areas of physical algorithms. Depending on the application and the considered time frame, we allow for many forms of network dynamics.

again, as researchers are starting to view the Internet and other computer science phenomena in the light of game theory.[3]

One of the cornerstones of game theory is the so-called equilibrium,[4] which describes a state of a system where no participant of the system has an incentive to change its strategy. As a perfect example of mathematicians mostly being interested in the question whether something can be done or not, pure game theorists are generally not interested in how such an equilibrium can be reached, or how long it takes to reach the equilibrium. Computer scientists on the other hand have been interested early on how to reach such an equilibrium [DGP06]. Clearly, this is an interesting playground for physical algorithms, as the participants of the systems cannot be modeled well by an input/output algorithm.

Today, game theory is a well-accepted subject in algorithms, probably the only one that does not follow the input/output paradigm that is well represented at theoretical computer science conferences.

Recently, research is starting to combine fault-tolerance with game theory [AAC+05].[5] In [MSW06], for instance, it is shown that the presence of Byzantine players may even contribute to the social welfare of a system.

2.2 Networks

Less known, the very same John von Neumann was also seminal in the development of the horizontal axis in our map of physical algorithms. Networks exist in many variants today, apart from the predominant Internet there are also niche areas such as sensor or peer-to-peer networks. More generally (and daring), one may even consider networks beyond computing, e.g. the human society or the brain.

Locality: All networks have in common that they are composed of a multiplicity of individual entities, so-called *nodes*; e.g. human beings in society, hosts in the Internet, or neurons in the brain. Each individual node can directly communicate only to a small number of neighboring nodes. On the other hand, in spite of each node being inherently "near-sighted", i.e., restricted to *local* communication, the entirety of the system is supposed to work towards some kind of *global* goal, solution, or equilibrium.

It is at the core of really understanding networks to know the possibilities and limitations of this *local computation*, i.e., to what degree local information

[3] Algorithmic game theory is a perfect example of the differences between the approach of computer science and mathematics/physics. In computer science, researchers try to understand the Internet by modeling the participants of the Internet as active (selfish) agents. Mathematicians and physicists take a more holistic approach and try to find random graphs that model all possible layers of the Internet (web pages, autonomous service providers, routers), or simply postulate that the bandwidth demands in the Internet are self-similar.

[4] Several different types of equilibria exist, e.g. Nash equilibria in games, or price equilibria in markets.

[5] One may argue that Byzantine agents are just selfish agents with a different goal.

is sufficient to solve global tasks [Lin92,Pel00,KMW04,Suo09]. Many tasks are inherently local, for instance, how many friends of friends one has. Many other tasks are inherently global, for instance, counting all the nodes of the system, or figuring out the diameter of the system. To solve such global problems, there is at least some information that must traverse long distances.

It is natural to ask whether there are tasks that are in the middle of these two extremes; tasks that are neither completely local nor inherently global. Indeed, this is the case. Assume for example that the nodes want to organize themselves, some nodes should be "masters", the others will be "slaves". The rules are that no two masters shall be direct neighbors, but every slave must have at least one master as direct neighbor. In graph theory, this problem is known as the *maximal independent set* (MIS) problem. Intuitively, this problem seems local since the rules are completely local. Consequently it might be expected that every node can communicate with its neighbors a few times, and together they can decide who will become master and who will become slave. However, this intuition is misleading. Even though the problem seems local, it cannot be solved using local information only! No matter how the system tackles the problem, no matter what protocol or algorithm the nodes use, non-local information is vital to solve the task. On the other hand, the problem is also not global: Mid-range information is enough to solve the problem. As such the MIS problem establishes an example that is neither local nor global, but in-between these extremes. Since at first sight it looks local, let us call it *pseudo-local*. Using *locality-preserving reductions* one can show that there exists a whole class of pseudo-local problems, similar to the class of NP-complete problems [KMW04].

This class of pseudo-local problems also includes many combinatorial optimization problems, such as minimum vertex cover, minimum dominating set, or maximum matching. In such problems, each node must base its decision (for example whether or not to join the dominating set) only on information about its pseudo-local neighborhood, and yet, the goal is to collectively achieve a good approximation to the globally optimal solution. Studying such local approximation algorithms is particularly interesting because it sheds light on the trade-off between the *amount of available local information* and the *resulting global optimality*. Specifically, it characterizes the amount of information needed in distributed decision making: what can be done with the information that is available within some fixed-size neighborhood of a node. Positive and negative results for local algorithms can thus be interpreted as information-theoretic upper and lower bounds; they give insight into the value of information [KMW06].

We believe that studying the fundamental possibilities and limitations of local computation is of interest to theoreticians in approximation theory, distributed computing, and graph theory. Furthermore, these results may be of interest for a wide range of scientific areas, for instance dynamic systems that change over time. The theory shows that small changes in a dynamic system may cause an intermediate (or pseudo-local) "butterfly effect," and it gives nontrivial bounds for self-healing or self-organizing systems, such as self-assembling robots. It also establishes bounds for further application areas, initially in en-

gineering and computing, possibly extending to other areas studying large-scale networks, essentially *physical algorithms*.

Studying locality and networks is at the heart of physical algorithms, as it is one of the few examples that have established some form of theory that uses concepts of complexity theory outside the input/output model.

Self-Organization and Dynamic Systems: Looking at the wider picture, one may argue that the idea of local algorithms as discussed in the last paragraph goes back to the early 1970s when Dijkstra introduced the concept of *self-stabilization* [Dij73]. A self-stabilizing system must survive arbitrary failures, including for instance a total wipe out of volatile memory at all nodes. The system must self-heal and eventually converge to a correct state from any arbitrary starting state, provided that no further faults occur.

It seems that the world of self-stabilization (which is asynchronous, long-lived, and full of malicious failures) has nothing in common with the world of local algorithms (which is synchronous, one-shot, and free of failures). However, as shown 20 years ago, this perception is incorrect [AS88], and the two areas are related. Intuitively, this is because (i) asynchronous systems can be made synchronous, (ii) self-stabilization concentrates on the case after the last failure, when all parts of the system are correct again, and (iii) one-shot algorithms can just be executed in an infinite loop. Thus, efficient self-stabilization essentially boils down to local algorithms and hence, local algorithms are the key to understanding fault-tolerance [LSW09].

Likewise, local algorithms help to understand *dynamic networks*, in which the topology of the system is constantly changing, either because of churn (nodes constantly joining or leaving as in peer-to-peer systems), mobility (edge changes because of mobile nodes in mobile networks), changing environmental conditions (edge changes in wireless networks), or algorithmic dynamics (edge changes because of algorithmic decisions in virtual or overlay networks). In dynamic networks, no node in the network is capable of keeping up-to-date global information on the network. Instead, nodes have to perform their intended (global) task based on local information only. In other words, all computation in these systems is inherently local! By using local algorithms, it is guaranteed that dynamics only affect a restricted neighborhood. Indeed, to the best of our knowledge, local algorithms always give the best solutions when it comes to dynamics. Dynamics also play a natural role in the area of self-assembly (DNA computing, self-assembling robots, shape-shifting systems, or claytronics), and as such it is not surprising that local algorithms are being considered a key to understanding self-assembling systems [Ste09,GCM05].

Social Networks: As already mentioned, there are numerous types of networks, including for instance the human society or the network of all the web pages in the world. Indeed, so-called *social networks* such as Facebook just merge the two concepts.

A decade ago Jon Kleinberg [Kle00] gave a first algorithmic explanation to a phenomenon studied almost a century ago. Back in the 1929, Frigyes Karinthy

published a volume of short stories that postulated that the world was "shrinking" because human beings were connected more and more. Some claim that he was inspired by radio network pioneer Guglielmo Marconi's 1909 Nobel Prize speech, to make the century complete. Despite physical distance, the growing density of human "networks" made the actual social distance smaller and smaller. As a result, any two individuals could be connected through at most five acquaintances, i.e. within six hops.

This idea has been followed ardently in the 1960s by several sociologists, first by Michael Gurevich, later by Stanley Milgram. Milgram wanted to know the average path length between two "random" humans, by using various experiments, generally using individuals from the US Midwest as starting points and from Boston as end points. The starting points were asked to send a letter to a well-described target person in Boston, however not directly, but only through an intermediate friend, hopefully "closer" to the target person. Shortly after starting the experiment, letters have been received. Often enough of course letters were lost during the process, but if they arrived, the average path length was about 5.5.

Statisticians tried to explain Milgram's experiments, by essentially giving network models that allowed for short diameters, i.e. each node is connected to each other node by only a few hops. Until today there is a thriving research community in statistical physics that tries to understand network properties that allow for "small world" effects. One of the keywords in this area are power-law graphs, networks were node degrees are distributed according to a power-law function.

This is interesting, but not enough to really understand what is going on. For Milgram's experiments to work, it is not sufficient to connect the nodes in a certain way. In addition, the nodes *themselves* need to know how to forward a message to one of their neighbors, even though they cannot know whether that neighbor is really closer to the target. In other words, nodes are not just following physical laws, but they make decisions themselves. In contrast to those mathematicians that worked on the problem earlier, Kleinberg [Kle00] understood that Milgram's experiment essentially shows that social networks are "navigable", and that one can only explain it in terms of a *greedy routing*.

In particular, Kleinberg set up an artificial network with nodes on a grid topology, plus one additional random link per node. In a quantitative study he showed that the random links need a specific distance distribution to allow for efficient greedy routing. This distribution marks the sweet spot for any navigable network. As such it is a great example for physical *algorithms*, because physical methods alone cannot find such a sweet spot, as statistical physicists hardly argue about algorithmic properties.

The are many applications for research in social networking, for instance viral marketing [KOW08], or spreading of biological viruses.

Physical Objects: A science fiction favorite are automatic cars equipped with distance sensors, following each other at high speed and minimal distance. This is a typical instance of physical objects organizing themselves. Unlike planets these

cars are not just following the laws of physics, but they will run algorithms that, depending on the values delivered by the distance sensors or other additional information (for instance wireless communication between cars), may speed up or slow down. Having studied control theory (or having been in a read-end collision accident) one knows that high speed lines of cars are not without problems. No matter what, the control loop will experience some delay. So if some car will need to break slightly, the next car might need to break a bit more already, and a few cars down the road we might have a terrible crash, because some car cannot break hard enough anymore.

There are several examples like this, e.g. coordination of helicopters or, more generally, any kind of swarm trying to maintain a certain formation based only on local distance information. In all these cases the agents may slowly drift from their desired (relative) position, because they are not able or willing to control their speeds perfectly. Furthermore, information on neighbors' positions could be outdated and/or inaccurate. Another example is clock synchronization in computer networks: Each node is equipped with a hardware clock which is not completely accurate, and nodes communicate local clock values by exchanging messages of varying delay with their neighbors.

All these examples have in common that they are algorithmic, in the sense that nodes have countless possibilities how to react to changes in the system. In a recent article [LLW10], tight bounds for some clock synchronization problem have been proved. These results are surprising in various ways. Even if all hardware clocks are accurate up to a few ticks per million ticks, it was shown that no matter what algorithm one uses, the error will depend on the size of the network. There is a simple algorithm that matches the lower bound using local information only, however, unfortunately, this algorithm is not intuitive. Indeed, it was shown that a number of canonical approaches are exponentially worse than the lower bound [LW06].

Real-time control of physical objects is of course beyond the input/output paradigm, still these problems usually have an algorithmic component which is worth studying. The range is large, from simple questions like how to organize a group of robot vacuum cleaners in order to clean a floor most efficiently, to the question of bird flocking [Cha09].

Wireless Communication: Network dynamics go well beyond mobility and failures. In some networks, communication is not graph- but geometry-based, in the sense that nodes can communicate with nearby nodes, only if not too many other nearby nodes transmit at the same time.

In the past, a large fraction of analytic research on wireless networks has focused on models where the network is represented by a graph. The wireless devices are nodes, any two nodes within communication (or interference) range are connected by an (annotated) edge. Such graph-based models are particularly popular among higher-layer protocol designers, hence they are also known as protocol models. Unfortunately, protocol models are often too simplistic. Consider for instance a case of three wireless communication pairs, every two of which can be transmitting concurrently without a conflict. In a protocol model one

will conclude that all three senders may transmit concurrently. Instead, in reality, wireless signals accumulate, and it may be that any two transmissions together generate too much interference, hindering the third receiver from correctly receiving the signal of its sender.

This many-to-many relationship makes understanding wireless transmissions difficult; a model where interference accumulates seems paramount to fully comprehend wireless communication. Similarly, protocol models oversimplify wireless attenuation. In protocol models the signal is usually binary, as if there was an invisible wall at which the signal immediately drops. Not surprisingly, in reality the signal decreases gracefully with distance. Because of these shortcomings, results for protocol models are often not applicable in reality. In contrast to the algorithmic ("computer science") community which focuses on protocol models, researchers in information, communication, or network theory ("electrical engineering") are working with wireless models that add up interference and take attenuation into account. A standard model is the *physical* model.

In the *physical model* the energy of a signal fades with distance. If the signal strength received by a device divided by the interfering strength of competitor transmitters (plus the noise) is above some threshold, the receiver can decode the message, otherwise it cannot. The physical model is reflecting the physical reality more precisely, hence its name. Unfortunately, most work in this model does not focus on algorithms with provable performance guarantees. Instead heuristics are usually proposed and evaluated by simulation. Analytical work is done for special cases only, e.g. networks with a grid structure, or random traffic. However, these special cases do neither give insights into the complexity of the problem, nor do they give algorithmic results that may ultimately lead to new distributed protocols. If one is interested in the capacity of an arbitrary wireless network, and how this capacity can be achieved, the community is not able to provide an answer.

However, this is about to change. Starting with [MW06], more and more algorithms work is adopting the physical model, thus combining the best of both worlds, by giving algorithms and limits for arbitrary wireless networks (not random node distributions), using the physical model (not the protocol model). We believe that bridging the gap between protocol designers and communication theorists is a fundamental challenge of the coming years, a hot topic for the wireless network community with implications for both theory and practice, and again a nice example of physical algorithms.

Multi-Core: Let us finish with what we started, multi-core systems. The switch to multi-core architectures promises increased parallelism, but not increased single-thread performance. Software developers are expected to handle this increased parallelism.

Today, the main tool for dealing with parallelism are *locks*; locks are software constructs that allow access to shared memory cells in a mutually exclusive way. However, there seems to be a general consensus in the computer science research community that locks are not the optimal programming paradigm to deal with concurrency and synchronization. Nobody really knows how to build large

systems depending on locks. The currently most promising solution is *transactional memory* [HM93]. Similarly to the database world, the programmer should encapsulate sequences of instructions within a transaction. Either the whole transaction is executed, or nothing at all. Other threads will see a transaction as one indivisible operation.

To some degree, the core of a transactional memory system is the *contention manager*. In case of a conflict between two transactions (e.g., both trying to store a value into the same memory cell), the contention manager decides which transaction must wait, or has to be aborted. The contention management policy of a transactional memory implementation can have a profound effect on its performance, and even correctness. Putting it simply, the contention manager inherits the role of the scheduler in a single core operating system.

In general, we believe that the key to understanding multi-core computing is through understanding networks. Transactions may for example be modeled as nodes in a *dependency graph* with edges between them in case of a conflict. Resolving conflicts based on local knowledge in this graph, solutions will scale canonically with multi-core systems growing. Thus, the problem naturally falls in the field of physical algorithms. A coloring of the graph yields a possible schedule for the transactions by executing all transactions of the same color in parallel.

An additional issue arises from the concurrent nature of programs in multi-core architecture. As concurrent processes interact and interfere with each other, processes also *compete* for some shared resources. If we keep in mind that in many cases, programmers who write code for a multi-core system are hardly interested in the performance of other processes, but merely on their own program's performance, we cannot desist from analyzing multi-core systems under the assumption that processes compete selfishly for system resources. To link back to our first example, game theory offers a great set of tools for this setting. We want to figure out whether existing multi-core systems are cheating-proof, i.e., incentive compatible in the sense that programmers have no interest to deviate from a behavior which is best for the overall performance of the system [EW09].

It is necessary to shed more light on the theoretical foundations of multi-core systems, with a special focus on transactional memory and its contention manager [SW09]. What one needs are refined models of efficiency and new contention managers that provably optimize the efficiency of transactional memory systems.

Acknowledgements: Thanks to Christoph Lenzen, for discussing the topic, and for reading the manuscript.

References

[AAC+05] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Michael Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *SOSP*, pages 45–58, 2005.

[ABBG10] Sanjeev Arora, Boaz Barak, Markus Brunnermeier, and Rong Ge. Computational complexity and informational asymmetry in financial products. Unpublished manuscript, 2010.

[AS88] Baruch Awerbuch and Michael Sipser. Dynamic networks are as fast as static networks (preliminary version). In *FOCS*, pages 206–220. IEEE, 1988.

[Cha09] Bernard Chazelle. Natural algorithms. In *Proc. of the 20^{th} ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2009.

[DGP06] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing (STOC)*, pages 71–78, New York, NY, USA, 2006. ACM.

[Dij73] Edsger W. Dijkstra. Self-stabilization in spite of distributed control. Manuscript EWD391, October 1973.

[EW09] Raphael Eidenbenz and Roger Wattenhofer. Good Programming in Transactional Memory: Game Theory Meets Multicore Architecture. In *20th International Symposium on Algorithms and Computation (ISAAC), Honolulu, Hawaii, USA*, December 2009.

[FLP83] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In *PODS*, pages 1–7, 1983.

[GCM05] Seth Copen Goldstein, Jason D. Campbell, and Todd C. Mowry. Programmable matter. *Computer*, 38(6):99–101, 2005.

[HM93] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. In *ISCA*, pages 289–300, 1993.

[Kle00] Jon M. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC*, pages 163–170, 2000.

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What Cannot be Computed Locally! In *Proc. of the 23^{rd} ACM Symposium on the Principles of Distributed Computing (PODC)*, pages 300–309, 2004.

[KMW06] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The Price of Being Near-Sighted. In *Proc. of the 17^{th} ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.

[KOW08] Jan Kostka, Yvonne Anne Oswald, and Roger Wattenhofer. Word of Mouth: Rumor Dissemination in Social Networks. In *15th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Villars-sur-Ollon, Switzerland*, June 2008.

[KPSS10] Fabian Kuhn, Konstantinos Panagiotou, Joel Spencer, and Angelika Steger. Synchrony and asynchrony in neural networks. In *Proc. of the 21^{st} ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[Lin92] Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[LLW10] Christoph Lenzen, Thomas Locher, and Roger Wattenhofer. Tight Bounds for Clock Synchronization. *J. ACM*, 57(2), 2010.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[LSW09] Christoph Lenzen, Jukka Suomela, and Roger Wattenhofer. Local Algorithms: Self-Stabilization on Speed. In *11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), Lyon, France*, November 2009.

[LW06] Thomas Locher and Roger Wattenhofer. Oblivious Gradient Clock Synchronization. In *20th International Symposium on Distributed Computing (DISC), Stockholm, Sweden*, September 2006.

[MSW06] Thomas Moscibroda, Stefan Schmid, and Roger Wattenhofer. When Selfish Meets Evil: Byzantine Players in a Virus Inoculation Game. In *25th Annual Symposium on Principles of Distributed Computing (PODC), Denver, Colorado, USA*, July 2006.

[MW06] Thomas Moscibroda and Roger Wattenhofer. The Complexity of Connectivity in Wireless Networks. In *25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Barcelona, Spain*, April 2006.

[Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[SPL80] R. Shostak, M. Pease, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. of the ACM*, 27(2):228–234, 1980.

[Ste09] Aaron Sterling. Memory consistency conditions for self-assembly programming. *CoRR*, abs/0909.2704, 2009.

[Suo09] Jukka Suomela. Survey of local algorithms. Manuscript, 2009.

[SW09] Johannes Schneider and Roger Wattenhofer. Bounds On Contention Management Algorithms. In *20th International Symposium on Algorithms and Computation (ISAAC), Honolulu, USA*, December 2009.

[vN93] John von Neumann. First Draft of a Report on the EDVAC. *IEEE Ann. Hist. Comput.*, 15(4):27–75, 1993.

[vNM47] John von Neumann and Oscar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1947.