Applications of Algebraic Topology to Concurrent Computation

Maurice Herlihy
Nir Shavit

Editorial preface

All parallel programs require some amount of synchronization to coordinate their concurrency to achieve correct solutions. It is commonly known that synchronization can cause poor performance by burdening the program with excessive overhead. This chapter develops a connection between certain synchronization primitives and topology. This connection permits the theoretical study of concurrent computing with all the mathematical tools of algebraic and combinatorial topology.

This article originally appeared in *SIAM News*, Vol. 27, No. 10, December 1994. It was updated during the summer/fall of 1995.

Today, the computer industry is very good at making computers run faster: speeds double roughly every two years. Eventually, however (and perhaps as early as the turn of the century), fundamental limitations, such as the speed of light or heat dissipation, will make further speed improvements increasingly difficult. Beyond that point, the most promising way to make computers more effective is to have many processors working in parallel, the approach known as multiprocessing.

The hard part of multiprocessing is getting the individual computers to coordinate effectively with one another. As a typical coordination problem, if two computers, possibly far apart, both try to reserve the same airline seat, care must be taken that exactly one of them succeeds. Coordination problems arise at all scales in multiprocessor systems—at a very small scale, processors within a single supercomputer might need to allocate resources, and at a very large scale, a nationwide distributed system, such as an "information highway," might need to allocate communication paths over which large quantities of data will be transmitted.

Coordination is difficult because multiprocessor systems are inherently *asynchronous*: processors can be delayed without warning for a variety of reasons, including interrupts, preemption, cache misses, and communication delays. These delays can vary enormously in scale: a cache miss might delay a processor for fewer than ten instructions, a page fault for a few million instructions, and operating system preemption for hundreds of millions of instructions. Any coordination protocol that does not take such delays into account runs the risk that a sudden delay of one process in the middle of a coordination protocol may leave the others in a state where they are unable to make progress.

The need for effective coordination has long been recognized as a fundamental aspect of multiprocessor architectures. As a result, modern processors typically provide hardware mechanisms that facilitate coordination. Until recently, these mechanisms were chosen in an ad hoc fashion, but it is becoming increasingly clear that some kind of mathematical theory is needed if the implications of such fundamental design choices are to be understood.

In this article, we focus on some new mathematical techniques for analyzing and evaluating common hardware synchronization primitives. Aside from its inherent interest to the computer science community, we believe this work may be of interest to the mathematical research community because it establishes a (perhaps unexpected) connection between asynchronous computability and a number of well-known results in combinatorial topology.

In many multiprocessor systems, processors communicate by applying certain operations, called *synchronization primitives*, to variables in a shared-memory. These primitives may simply be reads and writes, or they may include more complex constructs, such as *test-and-set*, *fetch-and-add*, or *compare-and-swap*. The test-and-set operation atomically writes a 1 to a variable and returns the variable's previous contents. The fetch-and-add operation atomically adds a given quantity to a variable and returns the variable's previous contents. Finally, the compare-and-swap operation atomically tests whether a variable has a given value and, if so, replaces it with another given value.

Over the years, computer scientists have proposed and implemented a variety of different synchronization primitives, and their relative merits have been the subject of a lively debate. Most of this debate has focused on the ease of implementation and ease of use of the primitives. More recently, however, it has emerged that some synchronization primitives are inherently more powerful than others, in the sense that every synchronization problem that can be solved by primitive A can also be solved by primitive B, but not vice versa. This article describes the new conceptual tools that are making it possible to provide a rigorous analysis of the relative computational power of different synchronization primitives. This emerging theory could provide the designers of computer networks and multiprocessor architectures with mathematical tools for recognizing when problems are unsolvable, for evaluating alternative synchronization primitives, and for making explicit the assumptions needed to make a problem solvable.

Our discussion focuses on a simple but important class of coordination tasks called *decision problems.* At the start with such problems, processors are assigned private *input values* (perhaps transmitted from outside). The processors communicate by applying operations to a shared-memory, and eventually each process chooses a private *output value* and halts. The decision problem is characterized by (1) the set of legitimate input value assignments and (2) for each input value assignment, the set of legitimate output value assignments. For example, consider the following *renaming* problem: as input values, each processor is assigned a unique identifier taken from a large range (like a social security number). As output values, the processors must choose unique values taken from a much smaller range. (Renaming is an abstraction of certain resource allocation problems.)

To solve a decision problem, a processor executes a program called a *protocol.* Because processors are subject to sudden delays, and because halting one processor for an arbitrary duration should not prevent the others from making progress, we require that each processor finish its protocol in a fixed number of steps, regardless of how its steps are interleaved with those of other processors. Such a protocol is said to be *wait-free*, since it implies that no processor can wait for another to do anything.

23.1. Simplicial Complexes

A decision problem has a simple geometric representation. Assume we have $n+1$ processes, each assigned a different color. A processor's state before starting a problem is represented as a point in a high-dimension Euclidian space. This point, called an *input vertex*, is labeled with a process color and an input value. Two input vertices are *compatible* if (1) they have distinct colors and (2) there exists a legitimate input value assignment that simultaneously assigns those values to those processes. For example, in the renaming problem described earlier, input values are required only to be distinct, so two input vertices are compatible if and only if they have distinct colors and distinct input values. We join any two compatible input vertices with a line segment, any three with a solid triangle, and any four with a solid tetrahedron. In general, any set of k compatible input vertices spans an *input k-simplex* in k-dimensional space. The set of all possible input simplexes forms a mathematical structure, called a *simplicial complex.* We call this structure the problem's *input complex.*

The notions of an *output vertex*, *output simplex*, and the problem's *output complex* are defined analogously, simply replacing input values with output values. The decision problem itself is defined by a relation Δ that carries each input n-simplex to a set of output n-simplexes. This relation has the following meaning: if S is an input simplex, T is an output simplex, and the processors start with their respective input values from S, then it is acceptable for them to halt with their respective output values from T.

For example, consider the instance of the renaming problem in which three processors are assigned unique input values in some large range and must coordinate to choose unique output values in the range 0 to 3. Here, an output simplex is a triangle whose vertices are labeled with distinct colors and distinct input values in the range 0 to 3. There are $4 \cdot 3 \cdot 2 = 24$ distinct output triangles, and it is not difficult to draw them on a sheet of paper. The result, shown in Figure 23.1, is topologically equivalent to a torus.

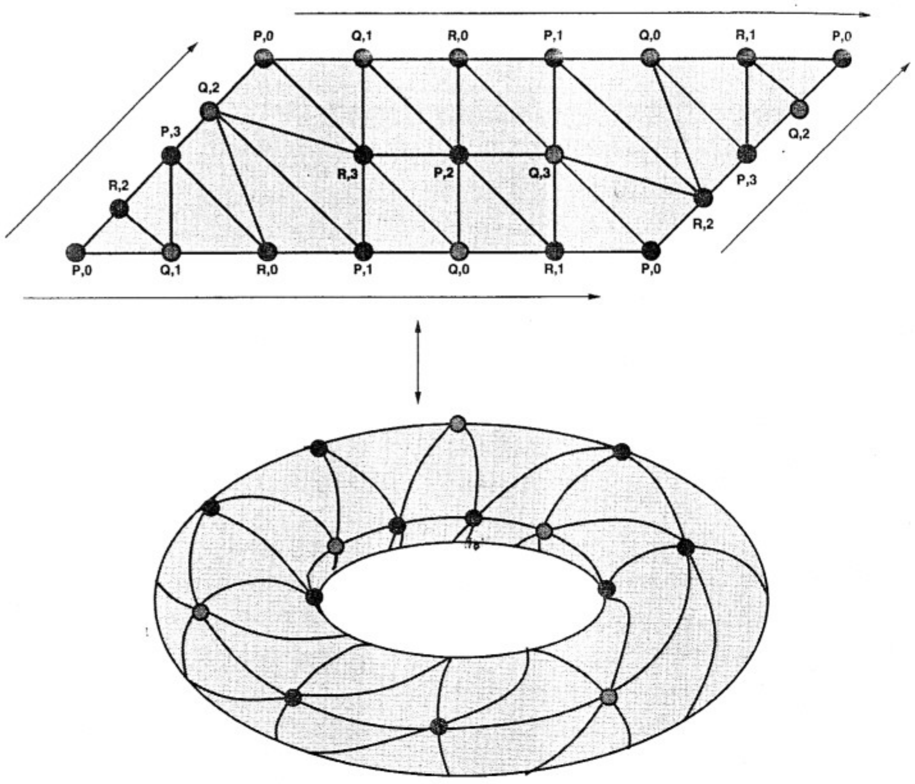

FIG. 23.1. *Three-process renaming with four names.*

Having shown how to specify a decision problem with a geometric model, we now do the same for the protocols that solve such problems. Recall that a protocol is a program: each processor starts out with its input value in a private register, applies a sequence of operations to variables in the shared-memory, and then chooses an output value based on the results of the computation. We can view any such protocol as accumulating a history of shared-memory operations—when the protocol has "seen enough," it computes its output value by applying a *decision map* to its history.

Any execution of a protocol generates a set of histories, one for each processor. The set of all possible executions also defines a simplicial complex: each vertex is labeled with a processor color and a history, and two vertices are compatible if they are labeled with distinct colors and if in some protocol execution, they see those two histories. We call this the *full-information complex* for the protocol. More precisely, for every input simplex S, any protocol induces a corresponding full-information complex $\mathcal{F}(S)$. The union of these complexes is the full-information complex for the protocol.

What does it mean for a protocol to solve a decision problem? Recall that a *decision map* δ carries each history h to the output value chosen by the protocol after observing h. The decision map induces a map from the full-information complex to the output complex: $\delta(\langle P, h\rangle) = \langle P, \delta(h)\rangle$. We are now ready to give a precise geometric statement of what it means for a protocol to solve a decision problem: given a decision problem with input complex $\mathcal{I}$, output complex $\mathcal{O}$, and relation Δ, a protocol solves a decision problem if and only if, for every input simplex $S \in \mathcal{I}$ and every full-information simplex $T \in \mathcal{F}(S)$, $\delta(T) \subset \Delta(T)$.

This definition is simply a formal way of stating that every execution of the protocol must yield an output value assignment permitted by the decision problem specification. Roundabout as this formulation of this property might seem, it has an important and useful advantage. We have moved from an operational notion of a decision problem, expressed in terms of computations unfolding in time, to a purely combinatorial description expressed in terms of relations among topological spaces. It is typically easier to reason about static mathematical relations than about ongoing computations, but, more importantly, this model allows us to exploit classical results from the rich literature on algebraic and combinatorial topology.

To prove that certain decision problems cannot be solved by certain classes of protocols, it is enough to show that no decision map exists. We can derive a number of impossibility results by exploiting basic properties that any decision map must have. In particular, any decision map is a *simplicial map*: it carries vertices to vertices, but it also carries simplexes to simplexes. Simplicial maps are also *continuous*: they preserve topological structure. If we can show that a class of protocols generates full-information complexes that are "topologically incompatible" with the problem's output complex, then we have established impossibility. Conversely, if we can prove that the decision map exists, then we have shown that a protocol exists.

A complex has *no holes* if any sphere embedded in the complex can be continuously deformed to a point. (More technically, the complex has trivial homotopy groups.) It has *no holes up to dimension* d if the same property holds for spheres of dimension d or less. (Notice that when d is zero, this condition means the complex is connected.) For example, a two-dimensional disk (e.g., a plate) has no holes, and a two-dimensional sphere (e.g., a basketball) has no holes up to dimension one, because any loop (e.g., a rubber band) on the sphere can be deformed to a point. By contrast, a torus has no holes only up to dimension zero—it is connected, but not every 1-sphere (loop) placed on the surface can be deformed to a point.

23.2. Read/Write Protocols

The simplest interesting synchronization primitives are atomic reads and writes to variables in shared-memory. We recently used this simplicial model to give a complete combinatorial characterization of the decision problems that can be solved by read/write protocols [8].

The full-information complexes for read/write protocols have a remarkable property: for any input simplex S, the full-information complex $\mathcal{F}(S)$ has no holes. This property holds for any read/write protocol, no matter how many variables it uses or how long it runs. This property is a powerful tool for proving impossibility results. A careful analysis of the renaming problem shows that if there are fewer than $2n+1$ possible output values, then the output complex has a hole. Moreover, any decision map must "wrap" a particular sphere in the full-information complex around that hole in such a way that the image of the sphere cannot be continuously deformed to a single point. Because the full-information complex has no holes, however, that sphere can be continuously deformed to a point in the full-information complex. Because the decision map is continuous, the image of that sphere can also be contracted to a point, and we have a contradiction. The same kind of analysis shows that a variety of fundamental synchronization problems have no wait-free solutions in read/write memory.

This topological model also yields a "universal" algorithm that can be used to solve any problem that can be solved by a wait-free read/write protocol. Any decision problem can be considered as a kind of "approximate agreement" problem in which each processor chooses a vertex in the output complex, and the processors negotiate among themselves to ensure that all processors choose vertices of a common simplex. This problem, which we call "simplex agreement," provides a simple normal form for any decision task protocol.

We can combine these two notions to give a complete characterization of the decision problems that can be solved by wait-free read/write protocols. Because the exact conditions require some technical definitions beyond the scope of this article, the focus here is on the underlying intuition. A decision problem has a wait-free read/write protocol if and only if the relation Δ can be "approximated" by a continuous map on its underlying point set, in the following sense. Given the input complex $\mathcal{I}$, construct a new complex, $\sigma(\mathcal{I})$, by subdividing each simplex in $\mathcal{I}$ into smaller simplexes. If v is a vertex in $\sigma(\mathcal{I})$, define *carrier*(v) to be the smallest simplex in $\mathcal{I}$ that contains v. The decision problem is solvable in read/write memory if and only if there exists a subdivision $\sigma(\mathcal{I})$ and a simplicial map $\mu : \sigma(\mathcal{I}) \rightarrow \mathcal{O}$ such that for each vertex $v \in \sigma(\mathcal{I})$, $\mu(v) \in \Delta(\mathit{carrier}(v))$. Informally, this condition states that it must be possible to "stretch" and "fold" the input complex so that each input simplex can cover its corresponding output simplexes.

This condition is shown schematically in Figure 23.2. The top half of the figure illustrates the relation Δ for a generic decision problem, and the bottom half shows how Δ can be approximated by a simplicial (continuous) map μ.

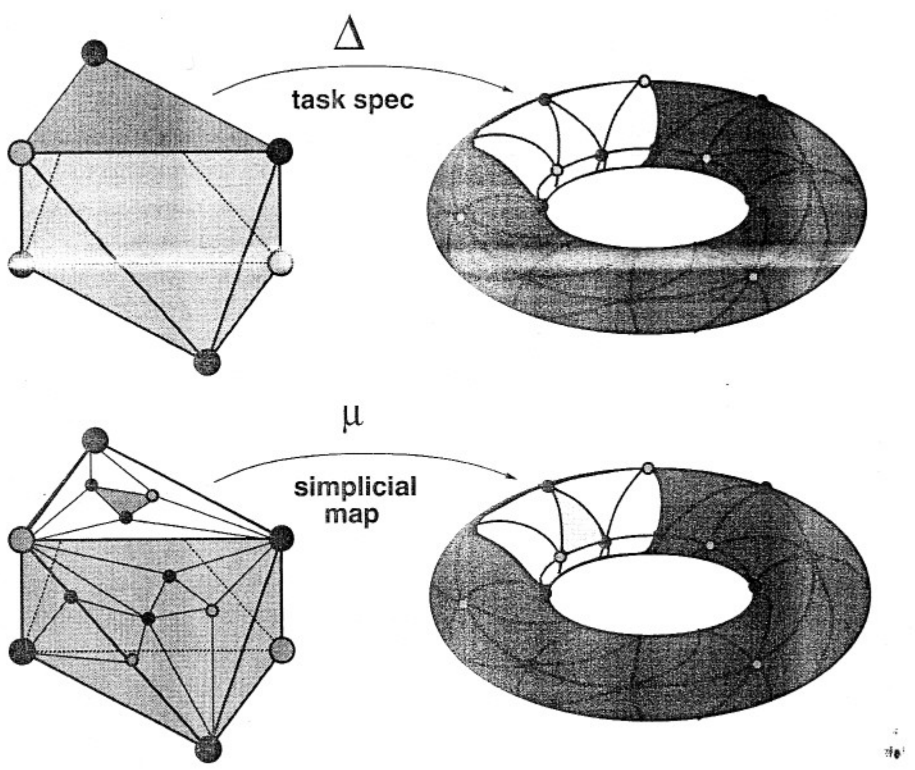

FIG. 23.2. *Existence condition for read/write protocols.*

23.3. Other Kinds of Protocols

Although read/write protocols have considerable theoretical interest, real multiprocessors typically provide more powerful synchronization primitives. The topology of full-information complexes for such protocols is more complicated. For example, Figure 23.3 shows the full-information complexes for two simple protocols in which processors communicate by applying test-and-set operations to shared variables. Casual inspection shows that these full-information complexes differ from their read/write counterparts in one fundamental respect: they have one-dimensional holes. Nevertheless, they do resemble them in another respect: they are connected. In general, any protocol in which $(n+1)$ processors communicate by pairwise sharing of test-and-set variables has a full-information complex with no holes up to dimension $\lfloor n/2 \rfloor$.

In a recent paper, Herlihy and Rajsbaum [6] analyzed the topological properties of full-information complexes for a family of synchronization primitives called *k-consensus* objects, which encompasses many of the synchronization primitives in use today. The larger the value of k, the more powerful is the primitive. The full-information complex for any protocol in which processes communicate via k-consensus objects has no holes up to dimension $\lfloor n/k \rfloor$. So at one extreme, when $k = 1$, the complex has no holes at all, and at the other extreme, the complex becomes disconnected. As k ranges from 1 to $n + 1$, holes appear first in higher dimensions and then spread to lower dimensions. A surprising implication of this structure is that there exist simple synchronization primitives that are *incomparable*: it is impossible to construct a wait-free implementation of one from the other.

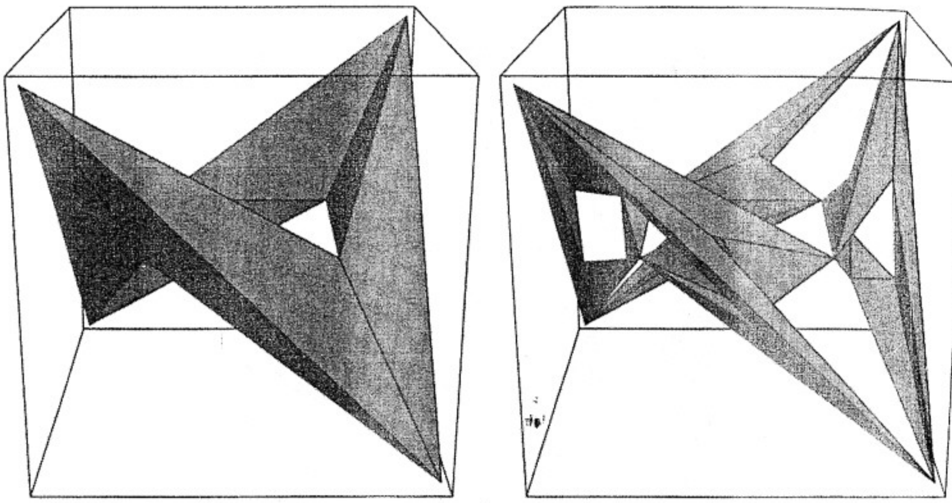

FIG. 23.3. *Full-information complexes for some test-and-set protocols.*

23.4. Related Work

The *consensus* problem is an idealized form of the transaction commitment protocols commonly used in distributed databases. In 1985, Fischer, Lynch, and Paterson [5] showed that if processors communicate by exchanging messages, then any consensus protocol has a "window of vulnerability" during which the failure or delay of a single processor will cause the protocol itself to fail or delay. This result showed that the notion of "asynchronous computability" differs in important ways from conventional notions of computability. Since then, a variety of research efforts have focused on characterizing the decision problems that can be solved by particular synchronization primitives in the presence of unpredictable failures and delays.

In 1988, Biran, Moran, and Zaks [1] gave a graph-theoretic characterization of decision problems that can be solved in the presence of a single failure in a message-passing system. This result was not substantially improved until 1993, when three independent research teams—Borowsky and Gafni [2, 3], Saks and Zaharoglou [9], and Herlihy and Shavit [7]—succeeded in applying combinatorial techniques to protocols that tolerate delays by more than one processor.

We provided the complete characterization of read/write solvability in two recent papers [7, 8]. The analysis of the topological properties of full-information complexes for protocols using more powerful primitives appears in a recent paper by Herlihy and Rajsbaum [6]. Recently, Chaudhuri et al. [4] were able to use similar topological techniques to derive the first lower bounds for a class of decision problems in a message-passing system in which processors execute in lockstep, but in which a processor can fail at any time by halting.

23.5. Conclusions

We believe this topological approach has a great deal of promise for the theory of distributed and concurrent computation, and that it merits further investigation. It has already produced a number of new and unexpected results and has illuminated an unexpected connection between the emerging theory of concurrent computation and the well-established theories of algebraic and combinatorial topology.

References

[1] O. Biran, S. Moran, and S. Zaks, *A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor*, in Proc. 7th Annual ACM Symposium on Principles of Distributed Computing, August 1988, pp. 263–275.

[2] E. Borowsky and E. Gafni, *Generalized flp impossibility result for t-resilient asynchronous computations*, in Proc. 1993 ACM Symposium on Theory of Computing, May 1993.

[3] E. Borowsky and E. Gafni, *Immediate atomic snapshots and fast renaming*, in Proc. 12th Annual ACM Symposium on Principles of Distributed Computing, August 1993.

[4] S. Chaudhuri, M. Herlihy, N. Lynch, and M.R. Tuttle, *A tight lower bound for k-set agreement*, in Proc. 34th IEEE Symposium on Foundations of Computer Science, October 1993.

[5] M. Fischer, N.A. Lynch, and M.S. Paterson, *Impossibility of distributed commit with one faulty process*, J. Assoc. Comput. Mach., 32(1985), pp. 374–382.

[6] M.P. Herlihy and S. Rajsbaum, *Set consensus using arbitrary objects*, in Proc. 13th Annual ACM Symposium on Principles of Distributed Computing, August 1994.

[7] M.P. Herlihy and N. Shavit, *The asynchronous computability theorem for t-resilient tasks*, in Proc. 1993 ACM Symposium on Theory of Computing, May 1993.

[8] M.P. Herlihy and N. Shavit, *A simple constructive computability theorem for wait-free computation*, in Proc. 26th Annual Symposium on Theory of Computing, May 1994, pp. 243–252.

[9] M. Saks and F. Zaharoglou, *Wait-free k-set agreement is impossible: The topology of public knowledge*, in Proc. 1993 ACM Symposium on Theory of Computing, May 1993.