DOI: http://dx.doi.org/10.7551/978-0-262-31709-2-ch072

Cooperation, Congestion and Chaos in Concurrent Computation

Mizuki Oka[1], Takashi Ikegami[2], Alex Woodward[2], Yiqing Zhu[3], Kazuhiko Kato[1]

[1]University of Tsukuba, Ibaraki 305-8577 mizuki,kato@cs.tsukuba.ac.jp
[2]University of Tokyo, Tokyo 153-8902, ikeg,alex@sacral.c.u-tokyo.ac.jp
[3]University of Tsukuba, Ibaraki 305-8577 sbbird.zhu@osss.cs.tsukuba.ac.jp

Abstract

We are interested in understanding how conflicts for common resources can be resolved when concurrently selfish agents are in place. To answer this question, we investigate a many-core machine that performs concurrent operations. Even with the selfish and non-cooperative nature of computational processes, they successfully organize a whole task. More specifically, we use the almost lock-free (ALF) architecture, which enables effective concurrent computation on a many-core machine. A unique point of the ALF is that it performs operations on shared resources simultaneously without excluding each other. We conducted data management experiments by varying the different number of cores on a single machine and investigating the characteristic dynamics of when the highest performance is observed. We found that the temporal dynamics of the number of operations changes from noisy to bursty pattern at the optimal point. In other words, the optimal computation is found at the edge of chaos. We argue that species or agents that interact concurrently with others show chaotic behavior in a congestion sate, and the cooperative state is established in the chaotic state.

Introduction

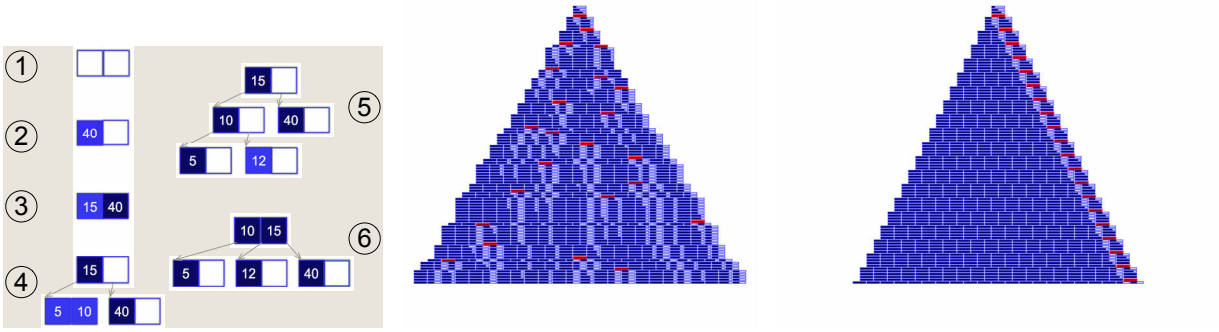
From multi-cellular organisms to swarms of birds and a large ecological system, there is a conflict for common resources, e.g., food, territories, etc. This type of conflict can be resolved by introducing temporal oscillation. When N number of agents can periodically access the resource in turn, a happy solution can be obtained where everybody can share an equal amount of the source. This periodic behavior is realized as turn-taking behaviors (Iizuka and Ikegami, 2004; Ikegami and Iizuka, 2007). Or the conflict can be resolved spatially by each agent sticking to its own niche (i.e., food/territory) without invading space belonging to others. This spatial division of niche is often observed in ecological system and other social systems.

But what happens if agents become selfish and access the resource at a time or invade the other niche? Does it always end up with an unhappy solution where nobody gets anything? Is it always bad manners to steal another's niche? In this paper, we investigate an artificial system that performs concurrent operations on many cores to answer these questions. More specifically, we are interested in understanding how parallel processing threads cooperatively work together and organize an entire task. We tackle this problem by having a new computational framework called "almost lock-free" (ALF for short), where we let each thread access a common work space without completely prohibiting others to access the same work space at the same time. A small interfering behavior will lead to an optimal behavior as we will show below.

ALF, presented here is a new algorithm we have invented for processing data concurrently in a computer with many cores (Wei and Kato, 2013). Due to the selfish and non-cooperative nature of computational processes, it usually is difficult to increase the throughput when performing concurrent operations on many cores. This is because, in order to maintain consistency of computation, *lock-operations* should be performed every time an operation accesses the common workspace. However, these lock operations become overhead and throughput decreases. Thus, when dealing with concurrent operations or multi-threading computation on many cores, how to process *lock-operations* are important factors to consider for achieving high throughput. ALF deals with this issue by permitting the mutual interference among processors. Using the ALF system, we will observe how concurrency causes congestions or conflicts, as well as how mutual cooperation emerges in such a system.

Almost Lock-Free System

One of the remarkable developments of processor industry in the last decade is the serial processing speed



(a) B-tree (order=2) (b) B-tree (order=4) with random keys (c) B-tree (order=4) with sorted keys

Figure 1: (a) An example of how B-tree (of order 2) is constructed. (b) An example of a large B-tree (of order 4) with randomly distributed key values. Colored space indicates the keys are inserted and uncolored space indicates the free space. Red node indicates the triggers of split. (c) An example of a large B-tree (of order 4) with ordered distributed key values.

or clock rate of core. Today, a standard computer is equipped with multi-core or even many-core processors. To make full use of these processors, concurrency control approaches have been proposed for writing concurrent programs. Dominant concurrency control approaches take what is called a pessimistic approach in which locks are performed every time a thread accesses the shared space. However, this extensive lock-based approach limits the concurrency of operations on multi-cores.

On the other hand, optimistic concurrency control approaches have been proposed. The optimistic approach assumes that multiple operations can complete without affecting each other. When conflict happens, the committed operations roll back. The optimistic approach can achieve a high throughput when conflicts are rare, since operations can complete without the expense of managing locks and without having operations wait for other operations' lock to clear. However, if conflicts happen often, the cost of restarting operations hurts performance significantly. ALF takes an approach that combines the pessimistic and optimistic concurrency control approaches, which we will explain in more detail below.

Balanced Tree Data Structure

Many different types of file systems exist such as HFS for Mac OS, Ext for Linux, NTFS for Windows machines, ISO 9660 used on DVDs and CDs and so on. They are different in directory structure, how much spaces files are allowed to use, what sort of metadata (about the usual data) is managed. But the basic purpose and architecture of modern file systems are similar to each other. In general, their purposes are managing access to the content of both data and the metadata available on local and global storage devices. In particular, a data structure called *balanced Tree* (B-tree for short) is used for organizing the indices in current file systems for efficiency; B-tree supports operations such as searches, insertions, and deletions in logarithmic time efficiency. ALF uses this B-tree data structure for managing concurrent operations.

B-tree is constrained to have an equal number of pointing nodes per each node in a data-address tree. A dynamic way of constructing B-Tree under this constraint provides a unique growth of the tree form. For example, Figure 1 shows how a B-Tree grows for an input sequence of key values (40, 15, 5, 10, 12). In the example, each node can contain two values, or order of 2. A value is inserted into a node in an ascending order until the node becomes full (step 1 to 3). When the node is full, it creates two child nodes as depicted in step 4 taking the median as the parent node. Then it continues to add a value to the tree in an ascending order. It happens that the tree becomes unbalanced as in the case of step 5. When this happens, the tree adjusts itself to make it balanced by moving an adequate value to its parent node as depicted in step 6. The nodes at the bottom of the tree are called leaf nodes and the other nodes are called internal nodes.

Properties on B-tree have been extensively studied

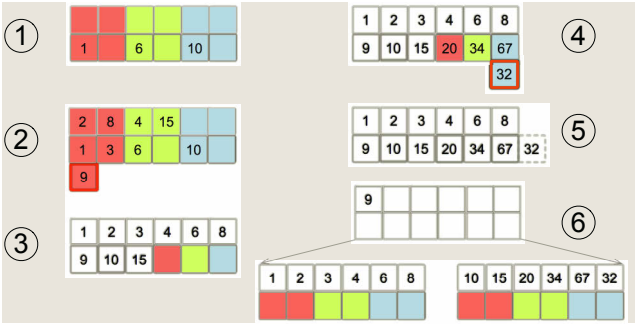


Figure 2: An example of how B-tree (of order 12) grows using ALF with three threads.

in the literature and the best-known property of B-tree is that it uses $log_e(2) = 69\%$ of spaces in each node when randomly distributed keys are inserted in B-tree (Johnson and Dennis, 1989). Examples of large B-trees are shown in Figure 1-(b) and -(c), with randomly distributed keys insertion and with sorted keys, respectively. These large B-trees show how leaf nodes are added or split with time. ALF uses this B-tree data structure for managing concurrent operations (i.e., insertion, deletion and searches).

ALF on B-tree

ALF takes a hybrid approach which combines optimistic and pessimistic concurrency controls on a B-tree data structure. More precisely, ALF operates on optimistic concurrency control and only executes locks when a certain condition is met. This may cause inconsistency during the data management processes. ALF achieves this by modifying the node in the data structure and allows many threads to update the same leaf node simultaneously. A key feature to combine optimistic concept with pessimistic concurrency controls is that it gives a minimum modification for tree node structures and concurrency controls. The concept of ALF is not just about managing the data, but it also tells us how each agent should behave independently and cooperatively with a common resource.

More practically, data structure is divided into two types, *public space* and *private space*. Each core operates on private space and interacts with each other through the public space. Since operations conducted on private space are not shared on the public space, other cores cannot see even if some operations can cause conflicts on the public space. For example, an insert operation conducted on a private space is not recognized by other cores until data is merged in the public space. If we lock the public space every time an operation affects the public space, or if we have a global clock, this conflict can be avoided.

Instead, ALF does not perform the lock every time an operation is performed on the public space, but rather lets it run until a certain condition is met, allowing some inconsistency in the data to occur. Without a global clock or complete lock operation, one may expect that the system will not self-organize anything due to the conflicts among private cores. Here we will show that this is not quite the case but rather it shows better throughput.

Figure 2 shows an example of how a B-tree grows with ALF. Here we take three threads as an example. Each thread corresponds to a core. The tree node structure is divided into two areas; the public space and the private space. The ALF adopts the idea of a thread-local area where threads can write to the private area simultaneously, to achieve a partial lock-free status. The data in the private space will be reflected on the public space by using exclusive locks when the private space becomes full. This operation is called **reorganization** and it happens when the private space for one thread becomes full. This reorganization phase is the only lock phase in the approach, and thus it is called almost lock-free.

Here, we explain how ALF works on B-tree (of order 12) by following the steps depicted in Figure 2. The initial node is assigned to private spaces and the same amount of space is allocated for each thread. Thread 1 is colored in red, thread 2 is colored in yellow and thread 3 is colored in blue, respectively. In this example, we only consider *insert* operations. We explain each step below.

1) Keys 1 (thread 1), 6 (thread 2), and 10 (thread 3) are inserted in each private region.

2) When the key 9 is assigned to thread 1 after inserting the key 2 in the same thread, it detects that the private space for thread 1 is full. This triggers reorganization of the node.

3) When reorganization is triggered, all the operations in the private area are reflected on the public area and keys are inserted in the public area in an ascending order. The remaining space is distributed equally for each thread.

4) Keys 20 (thread 1), 34 (thread 2), 67 and 32 (thread 3) are inserted on the private area. When the key 32 is inserted in thread 3, it triggers the reorganization again.

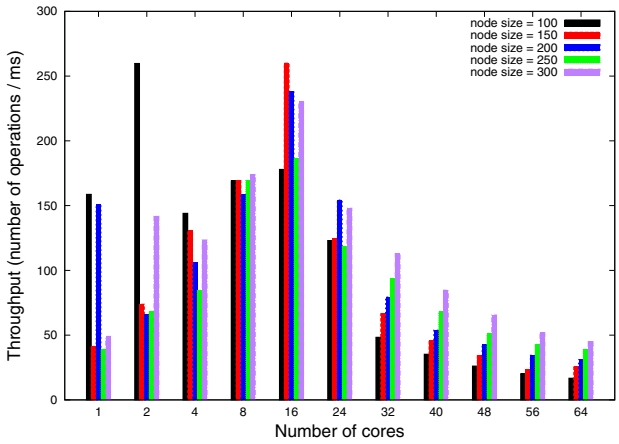


Figure 3: Throughput (number of operations executed per millisecond) when varying the number of cores and the orders.

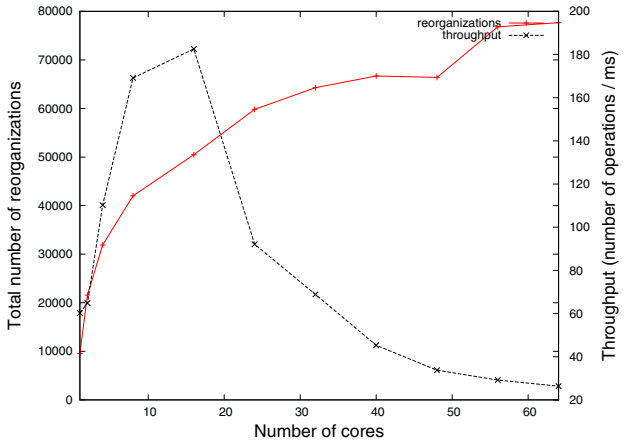


Figure 4: Total number of reorganizations and throughput with different number of cores. The throughput is maximized at 16 cores and after that the throughput decreases as the number of reorganizations increases.

5) All the keys in the private regions are inserted into the public area. However, the public area does not have enough space for all the keys in the private area. This triggers the node to split.

6) The medium key is taken as a parent node and two children nodes are created.

Note that private spaces are only allocated at the leaf nodes and all the internal nodes are allocated as public spaces.

Experiments

We conducted experiments using the ALF on 64-cores machine. Here, we only use insertion as operation. The total number of keys which are randomly manipulated is 1,000,000 of the range $[0, 1000000)$. The order (i.e., the node size) is set to 100, 150, 200, 250 and 300. We measure the total execution time on manipulating million keys by invoking system call [1], and then calculate the throughput (= number of operations per millisecond).

Best Degrees of "almost"?

Figure 3 shows the results of the throughput. The average throughput of the five runs is depicted in the figure. We see that the optimal throughput is obtained when the number of cores is 16 for all the orders and it decreases after that. This is because, although we gain a lot of concurrency, when the number of threads becomes larger, it also triggers a lot of reorganizations and thus leads to decrease in throughput. For example, in the case of 64 cores with the order of 150, only 2 or 3 spaces are allocated for each thread; resulting in the large number of reorganizations to occur. This is confirmed in the Figure 4, which plots the number of reorganizations and the throughput in relation to the number of cores (the order is set to 150). The number of reorganizations increases as the number of cores grows.

[1]The system call named CLOCK GETTIME is used.

Characteristic Dynamics of ALF

A characteristic feature of ALF, comparing with the extensive locking system, is that a larger number of reorganizations on B-tree can occur at a time as the number of threads becomes larger. The reorganization occurs when no space exists for executing operations on any thread in the private space.

Figure 5 shows the time evolution of the internal nodes and the free space ratio at each reorganization. Characteristic dynamics is observed in the stepwise increases of the number of internal nodes. The reorganization makes new space for each thread to execute the operations and the number of free space will increase following the number of internal nodes of a tree. Having too many cores leads to frequent reorganization, slowing down the entire performance.

On the other hand, if it has enough space, many cores can work concurrently, increasing the entire perfor-

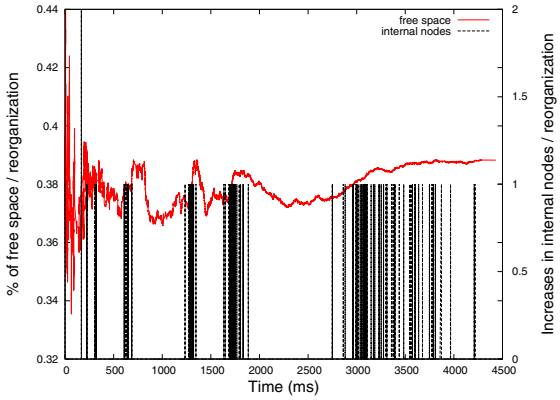
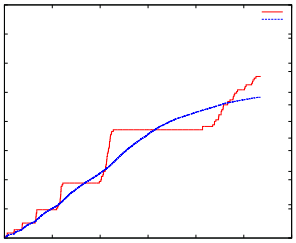
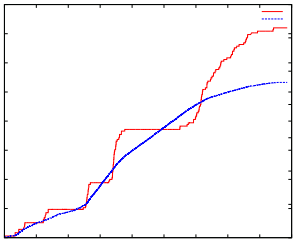


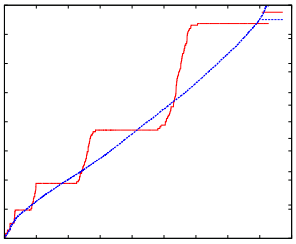
Figure 5: Dynamics of changes in the ratio of the free space in the entire B-tree and the increase in the number of internal nodes.



(a) cores = 4



(b) cores = 16



(c) cores = 64

Figure 6: Time evolution of the total number of leaf nodes and internal nodes of the B-tree. The red colored line shows the total number of internal nodes and the blue colored line shows the total number of leaf nodes.

mance. As we saw in Figure 3, the maximum throughput is found around when the number of core equals to 16. If operation load is equally balanced among concurrently processing threads, the number of possible operations should be proportional to the number of cores. But actually, unbalanced operational loads occur that suppresses the number of operations resulting in the decrease of throughput.

In Figure 6, the number of leafs and internal nodes develop differently in time depending on the number of cores. For the number of cores equal to 4, the number of leafs shows a convex curve, whereas that of 64 shows the concave and that of 16 is hybrid. For the number of internal nodes, all the examples show stepwise development, except that the case of the number of cores equals to 16, the step size does not grow geometrically, but rather with some modulations. These are the pieces of circumstantial evidences that the core number equal to 16 is at the boundary of two quantitatively different dynamics phases.

The singularity of the core = 16 is also reflected in the time evolution of the free space ratio and the number of operations. Figure 7 shows the dynamics of the number of operations and the ratio of free space of the entire B-tree. The leftmost figure corresponds to the single core case, and the right most figure corresponds the case with 64-cores. When the number of cores is below 16, the possible number of operations over a course of time is suppressed at a lower value around 100. By further increasing the number of cores, the number of operations will be raised to around 500 with bursty time series. The critical core number 16 corresponds to the transition point. To confirm this transition, we counted the number of local peaks in the time series of the number of operations 8. We superimpose all the extracted peak values by changing the number of cores. We can observe a quantitative transition when the number of cores is 16.

We can summarize the behavior of the number of operations and of free space as follows.

i) The number of operations can be classified into two patterns: a noisy time series with lower amplitudes and a bursty time series with larger amplitudes. The

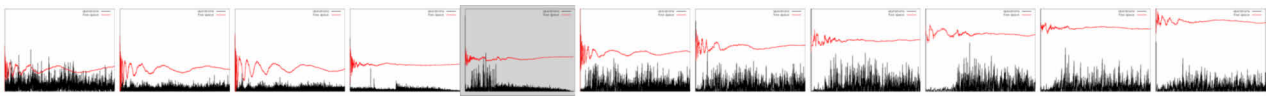


Figure 7: Dynamics of the number of operations and the ratio of free space of the entire B-tree. The figures are shown for cores = 1, 2, 4, 8, 16, 24, 32, 40, 48, 56, and 64 from left to right. The optimal throughput is found at core = 16 and is colored in grey.

optimal throughput (core = 16) is found at the transition point of these patterns. An exception is the single-core case whose time evolution of operations is similar to the optimal case.

ii) The number of free space on average is proportional to logarithm of the number of cores.

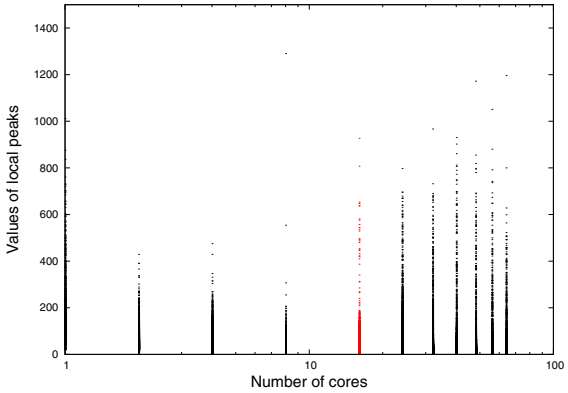


Figure 8: Counting the number of local peaks in the time series of the number of operations, we superimpose all the extracted peak values (y-axis) by changing the number of cores (x-axis). It should be noted that there is a qualitative transition at the number of cores equals to 16 (colored in red).

From these observations, we say that the optimal number of cores for the entire computation is found at the transition point, which is at the edge of the chaotic state and the bursty phase.

Discussions

We know several examples showing that the optimal behavior can be found at the edge of chaos. This paper adds another example that the optimal computation is found at the edge of chaos and the bursty behavior. In our previous work, we have also found that the optimal throughput of the packet switching network (PSN) at the edge of chaos and the periodic window (Ikegami et al., 2011; Takayasu, 2005). In the case of PSN, congestion of packets occurs at the critical point where the throughput becomes optimal. In an analogy with PSN, we hypothesize that congestion among different threads allows the system to perform more operations.

Concurrency causes congestion and congestion lets a system rearrange the B-tree structure, creating more free space. That is, with the increase in the size of free space, effective competition among threads is suppressed. In other words, a mutual cooperation emerges. A juxtaposition of three unrelated C-terms, concurrency, congestion and cooperation is linked by the dynamics at the edge of chaos.

A similar discussion can be applied to an ecological system's dynamics. Host and parasite networks organize a complex food web. With respect to the population dynamics of each species dynamics, we know that a weak chaos with large degrees of freedom, called *homeochaos* (Kaneko and Ikegami, 1992) leads to a network symbiotic state (i.e., cooperative phenomena). This chaotic state is attained by auto-tuning dynamics of mutation rates of each species. An initial set of species self-organizes into this homeo-chaotic state by increasing the mutation rates.

We believe that the biodiversity of a rainforest provides such an example. The abundance of each species in a rainforest is relatively low but many different species can co-exist in the same place (Connell, 1978). We argue that congestion of species produces chaotic dynamics in an ecosystem and they work concurrently. That is, species or agents that interact concurrently with others will show chaotic behavior in a congestion sate, and the cooperative state is established in the chaotic state. The current work provides that the same principle can be applied in a concurrent computing system.

Conclusion

We proposed a new idea of effective concurrent computation without using the scheme of extensively locking. A unique point of this scheme is that all the threads perform operations simultaneously without excluding each other. We found that the optimal number of cores for efficient computation is 16 in our experimental setting. The temporal dynamics of the number of operations changes from noisy to bursty pattern at the optimal point. We thus insist that the optimal computation is found at the edge of chaos. The emergence of this critical point comes from the almost lock scheme.

References

Connell, J. H. (1978). Diversity in tropical rain forests and coral reefs. *Science*, 199:1302–1310.

Iizuka, H. and Ikegami, T. (2004). Adaptability and diversity in simulated turn-taking behavior. *Artificial Life*, 10:361–378.

Ikegami, T. and Iizuka, H. (2007). Turn-taking interaction as a cooperative and co-creative process. *Infant Behavior and Development*, 30(2):278–288.

Ikegami, T., Oka, M., and Abe, H. (2011). Autonomy of the internet: complexity of flow dynamics in a packet switching network. In *Proc. of the 20th European Conference on Artificial Life*, pages 364–371.

Johnson, T. and Dennis, S. (1989). Utilization of b-trees with inserts, deletes and modifies. In *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '89, pages 235–246, New York, NY, USA. ACM.

Kaneko, K. and Ikegami, T. (1992). Homeochaos: dynamics stability of a symbiotic network with population dynamics and evolving mutation rates. *Physica D: Nonlinear Phenomena*, 56:406–429.

Takayasu, M. (2005). Dynamic complexity in the internet traffic. In Kocarev, L. and Vattay, G., editors, *Complex Dynamics in Communication Networks*, Understanding Complex Systems, pages 329–358. Springer Berlin Heidelberg.

Wei, C. and Kato, K. (2013). Study on concurrent almost lock-free b-tree operations. Master thesis at the University of Tsukuba.