STSM: Theory of Transactional Memory
ECOST-STSM-IC1001-010912-017912
Research Report

October 26, 2012

Disjoint-access parallelism and wait-freedom are two desirable properties for implementations of concurrent objects. *Disjoint-access parallelism* guarantees that processes operating on different parts of an implemented object do not interfere with each other by accessing common base objects. Thus, disjoint-access parallel algorithms allow for increased parallelism. *Wait-freedom* guarantees progress for each nonfaulty process, even when other processes run at arbitrary speeds or crash.

Transactional memory (`TM`) provides a general mechanism for obtaining a concurrent implementation of any object from its sequential code. `TM` supports the execution of *transactions*. A transaction tries to apply the sequential code of an operation supported by some object in a concurrent setting and it may either succeed, in which case we say it *commits*, or fail, in which case we say it *aborts*. If a transaction aborts, none of its changes ever become visible.

In [1], we have shown that there is no `TM` algorithm that ensures both disjoint-access parallelism and wait-freedom. More specifically, we have shown this impossibility result for `TM` algorithms that continuously restart a transaction whenever it aborts, until it commits. Moreover, in [1], in order to overcome this impossibility result we have presented a `TM` algorithm that ensures disjoint-access parallelism and wait-freedom when applied to objects that have a bound on the number of data items accessed by each operation they support.

1 Description of the work carried out during the STSM

The goal of this scientific mission is to discover whether our previous impossibility result holds for weaker definitions of disjoint-access parallelism, and if this is so, to find different ways to overcome it. These two goals where partially covered during the first visit of Alessia Milani to FORTH, in June 2012, where we had some preliminary ideas on both directions.

More specifically, we tried to extend the impossibility result by considering a weaker definition of disjoint-access parallelism, called *partial disjoint-access parallelism*. Intuitively, partial disjoint-access parallelism allows two processes executing a `TM` algorithm to interfere with each other on some specific set of base objects (e.g., on a single timestamp object), even when they operate on different parts of the implemented object. Formal definition of this property is provided in the next section. Then, we tried to prove that there is no `TM` algorithm that ensures both partial disjoint-access parallelism and wait-freedom. Although we have not completed this proof yet, we have a much better understanding of this proof's subtleties.

Recall that the algorithm, DAP-UC, we presented in [1], ensures disjoint-access parallelism and wait-freedom, when applied to objects that have a bound on the number of data items accessed by each operation they support. During the first visit, we designed a new algorithm, called PDAP-UC, that ensures partially disjoint-access parallel and wait-freedom, when applied to any object with bounded number of entry points. Intuitively, an *entry point* is a data item that is accessed first by an operation of the object.

During the second visit, we first introduced a new disjoint-access parallel property called *transitive disjoint-access parallelism*, or *tdap*. Intuitively, this property allows two transactions to contend on some base object even if they operate on different parts of the implemented object, when their execution intervals overlap with other transactions which are allowed to contend under (our original definition of) disjoint-access parallelism. Since, this property is stronger than disjoint-access parallelism, our previous result [1] is applied; hence, it follows that no TM algorithm is tdap and wait-free for unbounded objects. Therefore, we defined the *partially tdap* property, which is similar with partial disjoint-access parallelism except that disjoint-access parallelism is replaced with tdap. Formal definitions of these properties are provided in the next section.

Then, we designed a new algorithm, called PtDAP-UC, that is partially tdap and wait-free, for unbounded objects with unbounded entry points.

2 Description of the main results obtained

In this Section a preliminary version of the results obtained is presented.

2.1 Formal definitions of new properties

For simplicity, we use the term operation to refer to the execution of a transaction until it commits, given that the transaction is restarted whenever it aborts. Two operations *overlap* if one of them in invoked during the execution interval of the other. If a process has invoked an operation that has not yet returned, we say that the operation is *active*. A process can have at most one active operation in any configuration.

Disjoint-Access Parallel. We start by providing the definition of disjoint-access parallelism, as presented in [1].

Fix any execution $\alpha = C_0, \phi_0, C_1, \phi_1, \ldots,$ produced by a linearizable TM algorithm U. Then there is at least one linearization of the completed operations in α and some subset of the uncompleted operations in α such that the responses of all these operations are consistent. Fix any such linearization l. Let op be any one of these operations, let I_{op} be its execution interval, let C_i denote the first configuration of I_{op}, and let C_j be the first configuration at which op has been linearized in l. Since each process has at most one uncompleted operation in α and each operation is linearized in l, within its execution interval, the set of operations linearized before C_i is finite. For $i \leq k < j$, let S_k denote the state of the object which results from applying each operation linearized in α prior to configuration C_k, in order, starting from the initial state of the object. Define $DS(op, \alpha)$, the data set of op in α, to be the set of all data items accessed by op when executed by itself starting from S_k, for $i \leq k < j$.

The *conflict graph* of an execution interval I of α is an undirected graph, where vertices represent operations whose execution intervals overlap with I and an edge connects two operations op and op' if and only if $DS(op, \alpha) \cap DS(op', \alpha) \neq \emptyset$.

The *conflict graph induced by two* operations op_1 and op_2, denoted $CG(op_1, op_2)$ is the conflict graph of the minimal execution interval containing op_1 and op_2.

Definition 1 (Disjoint-Access Parallelism). *A* TM *algorithm is* disjoint-access parallel *if, for every execution containing a process executing an operation op_1 and a process executing an operation op_2 that contend on some base object, there is a path between op_1 and op_2 in $CG(op_1, op_2)$*

Transitively DAP. The new definition is a light variant of Definition 1. The main difference is on the way the conflict graph induced by two operations is defined. The new conflict graph includes slightly more operations than the conflict graph used to define our original disjoint-access parallelism definition, therefore allowing more contention on base objects.

The conflict graph $CG'(op_1, op_2)$ induced by two operations op_1 and op_2 is an undirected graph. Each vertex of $CG'(op_1, op_2)$ corresponds to an operation of the set $\mathcal{O}$ defined as follows.

Let

$$\mathcal{I} = \bigcup_{op : I_{op} \cap I_{op_1} \neq \emptyset \vee I_{op} \cap I_{op_2} \neq \emptyset} I_{op}$$

Then,

$$\mathcal{O} = \{op : I_{op} \cap \mathcal{I} \neq \emptyset\}$$

That is, $\mathcal{I}$ is the minimal execution interval that contains the execution intervals of op_1 and op_2 as well as the execution interval of every operation whose execution interval intersects the execution interval of op_1 or the execution interval of op_2. $\mathcal{O}$ is then the set of operations whose execution interval intersects $\mathcal{I}$. Finally, as in the definition of disjoint-access parallelism, two vertexes corresponding to two operations op and op' share an edge if and only if $DS(op) \cap DS(op') \neq \emptyset$.

Definition 2 (tdap)**.** *A* `TM` *algorithm is tdap if, for every execution containing a process executing an operation op_1 and a process executing an operation op_2 that contend on some base object, there is a path between op_1 and op_2 in $CG'(op_1, op_2)$*

Partial Disjoint Access Parallelism and Partial tdap. An implementation is *partially dap* if it is dap except for some base objects in a small set B. That is, for every base object $b \notin B$, two operation contending on b must be connected in the conflict graph induced by these two operations. On the contrary, two operations contending on some base object $b \in B$ are not required to satisfy any constraint.

Definition 3 (Partial disjoint-access parallelism)**.** *Let B be a set of base object. A* `TM` *algorithm is* partially disjoint-access parallel with respect to B *if it satisfies disjoint-access parallelism for any base object $b \notin B$.*

Definition 4 (Partial tdap)**.** *Let B be a set of base object. A* `TM` *algorithm is* partially tdap with respect to B *if it satisfies tdap for any base object $b \notin B$.*

In the following, the set B of base objects on which any operation may contend, is restricted to contain only a single timestamp object. For simplicity, we will say that an implementation is partially disjoint-access parallel or partially tdap instead of partially disjoint-access parallel or partially tdap with respect to a timestamp object.

2.2 PtDAP-UC

The new algorithm is based on the concept of a *chain of entry points*, which is defined as follows. Assuming that each operation starts using a single entry point, two data items participate to the same chain of entry points, if the one has been created during an operation that has started using as an entry point the other. Notice that the last element of each such chain is some static data item. Also, a single data item may participate to several such chains.

PDAP-UC fails to guarantee wait-freedom for unbounded entry points, because roughly speaking it "blocks" only entry points, while the new algorithm is able to "block" chains of entry points. Intuitively, when an operation op blocks an entry point e (or a chain of entry points), it means that some operation using e (or any data item participating in this chain) as an entry point and is invoked after this blocking occurred will help op complete, before its response. Specifically, a data item in PtDAP-UC contains as metadata a field *pentry* that describes the entry point through which it has been created, and some announce array where an operation can be announced and other operations can discover it and help it. Each operation op announces itself on each data item x it accesses.

Then, whenever op creates a data item x starting from some entry point y, it initializes x so that $x.pentry := y$. It is important that by doing this, all the elements participating x's chain of entry points can be accessed by continuously following the corresponding *pentry* pointers, starting from $x.pentry$, until a *pentry* pointer with value *null* is reached (which is the value of the *pentry* pointer of the last data item in this chain). So, in the new algorithm during the creation of x, op accesses all the data items participating to the same chain with x and on each `varrec` z it access, it helps those announced operations in $z.A$ with timestamp value smaller that the timestamp value of z.

The new algorithm ensures wait-freedom, since i) the set of created data items when some operation op starts its execution is bounded, and ii) after some point in time op will "block" all the chains of entry points created after its initiation. Moreover, we can prove that the new algorithm guarantees partially tdap.

3 Conclusion

The team continues the collaboration by currently working to finalize the formal proof of correctness for the new algorithms (PDAP-UC and PtDAP-UC). Moreover, we make efforts toward proving a new impossibility result (stating that no partially disjoint-access parallel TM algorithm is wait-free). We expect that the third and last visit during this scientific mission will significantly contribute to the completion of this work and its publication to some high quality conference.

References

[1] F. Ellen, P. Fatourou, E. Kosmas, A. Milani, and C. Travers. Universal constructions that ensure disjoint-access parallelism and wait-freedom. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, PODC '12, pages 115–124, Madeira, Portugal, 2012.