Summary:

The paper offers an approach, combining traditional Software Transactional Memory (STM) systems with a Machine Learning (ML) algorithm, which enables the selection of the best-performing STM algorithm dynamically at run-time.
STM algorithms differ in various implementation details, e.g. how conflicts between writers and readers are detected (eager/lazy conflict detection scheme) and which internal data structures are used therefore. All these implementation details influence how well certain programs perform using a given STM algorithm, it is even possible that certain pathological situations, which could lead e.g. to starvation, can be avoided by switching the algorithm. Therefore STM implementations need a way to adapt, if they should deliver peak performance. The adaption can even depend on certain environment factors of the running application.
The presented approach combines an offline learning, which tries to estimate the characteristics for all given algorithms, with dynamic profiling to select the best fit at run-time. The evaluation characteristics for a given STM algorithm include the different accesses to shared memory, the "time gap" between successive transactions, in order to estimate the non-transactional work per transaction and the write frequency, since read-only transactions can be executed with a minimal overhead on most STMs. During the initial offline learning all algorithms are classified on a series of micro-benchmarks, which should cover a broad variety of common conditions in a STM system. Therefore two different ML approaches are used: case-based reasoning (CBR) and neural networks with the NEAT algorithm. The learned characteristics are finally expressed as a so called *policy*.
Finally, during execution of the program, the policy is consulted to select an appropriate algorithm after measuring N transactions. A re-evaluation of the current algorithm is triggered during run-time only by three events: on each creation of an new thread, if a thread aborts more than 16 consecutive times, or when a thread blocks more than 2048 cycles, when trying to start a transaction.

Towards Applying Machine Learning to Adaptive Transactional Memory

1. Summary

This paper discusses about how to choose among available STM algorithms by building profiling information at runtime, and also using machine learning techniques on these profiling information. Each STM algorithm has different efficiency for different platforms and workloads. Proposed system uses profiling information and learning algorithms to choose the STM that best suits for the platform and given workload.

System distinguishes the workload based on many criteria. Firstly, shared memory access types. There are five such types discussed based on whether access is read or write, or whether it happens after or before the first write and so on. Second criteria is the amount of non-transactional work involved in the workload. Another criteria discussed is the writer frequency in the workload. For each such workload, there is no single STM which works best for all the above.

Next the author discusses about building profiling information for different STM algorithms. A runtime system called profileTM is used that prevents concurrent execution using some kind of locks. ProfileTM also uses a hardware tick counter to measure the execution time of different transactions. There is no global metadata accesses. All these ensure that transactions are executed fast on profileTM.

Initially there is an offline training in which all STM algorithms are run and an adaptive policy is built on some benchmarked data. Authors used many benchmark configurations like Red-black trees, linked lists etc on 15 different STM algorithms. Each of these workload has different configurations. During training such data, system builds case bases which represent learning of that training. Such learnt data is used at runtime to choose the appropriate STM algorithm. Instead of these case bases, neural networks can also be used for learning which is more complex than case based reasoning.

2. Open Questions

1. In section 3.2, it is mentioned that profileTM has to wait for completion of all the in-flight tasks and also for the completion of the tasks that are waiting for locks. Can this be achieved without any changes to scheduler?

Towards Applying Machine Learning to Adaptive Transactional Memory

Summary

This weeks paper discusses the application of Machine Learning (ML) in order to adopt Software Transactional Memory (STM) systems by switching dynamically between different implementations.

Similar to other approaches, they have an evaluation mechanism which, based on certain measurements for throughput, tries to find the best STM implementation by trying out several implementations online. The system is supplied with the results of a previous off-line training for all algorithms available. This training is based on a fixed set of microbenchmarks. It is worth mentioning that they need to alter the implementation of each STM implementation (at least slightly) in order to get their evaluation data which seems to be one pitfall of the system they didn't mention.

What is new in their approach is the use of machine learning. They are evaluating two different approaches. One is Case-Based Reasoning, which seams to be a kind of pattern matching based on a set of learned base cases (derived during off-line training?). The other one is based on neural networks, called NEAT (Neuro Evolution of Augmenting Topologies), which tries to find a mathematical function (model?) that covers all cases of the training data. How these methods are used to optimize the system is not totally clear to me, though.

Nevertheless, it seems to work, given the evaluation results which occur to be neither that bad in average, nor as good as desired, which is jutified by the authors with the early state of the work. There are several open problems such as a more complete and overall better microbenchmark sets as well as additional measurements for the selection mechanism.

Open Questions

- What are orecs?
- To me CBR looks more like pattern matching then optimization. Where does the algorithm compare solutions and find the best one?
- Why are they knowingly using an incomplete set of training microbenchmarks (see 5.8)? (...Ok, the conclusion refers to that partially, but nevertheless.)

Summary Machine Learning

The paper presents a system which can choose between multiple STM systems at runtime to adapt to the current workload using machine learning. The authors justify the relevance of this by arguing that different STM algorithms are suited for different workloads (e.g. high conflict rates versus mostly read only transactions). They also mention that programs often have multiple "phases" with distinct characteristics. Even if one STM algorithm outperforms all others on such a program, it might still make sense to switch to a different implementation that is more suitable for specific phases. In order to determine an optimal STM, the authors first present the features that they measure to base their choice upon. They measure shared memory accesses (sequence of read and write operations), the time spent sequentially between running transactions and the write frequency. They explicitly mention that they do not measure the abort rate, claiming that it does not correspond to throughput. They also avoid mesauring I/O operations, as those do not occur in all workloads. In order to actually measure the meory accesseses, the authors employ a runtime system (ProfileTM). It seems to run transactions sequentially. The system first does some off-line training on benchmarks and use machine learning to extract relations between workloads and optimal STM algorithms from those. This is done only once. When a programm is started, the system chooses an initial STM algorithm (depending on required program semantics like ELA). Furthermore, the system can readabt when encountering specific events. Those triggers are consecutive aborts, newly created threads and stall periods. In those cases, ProfileTM is run to collect information about the current workload. Afterwards, the system uses the data to choose a (potentially) new STM algorithm, in the hope that it performs better.

As already stated, the authors employ machine learning to find the best algorithm. They tried to different approaches, case-based reasoning and neural networks. Case-based reasoning uses a data set of previously solved issues to reason about a new problems, matching it to the most similiar already encountered issue. Neuronal networks on the other hand derive a function mapping input (here: the workload) to the obsvered output. Then, at runtime the function is simply applied to the new input. While CBR is more expensiave at runime, the authors' evaluation showed that it results in larger speedups.

In their evaluation the system was also compared against an oracle, which chose the best, single STM algorithm. Thanks to the adaption, the system could sometimes deliver a better performance than the single algorithm. On the other hand, in some cases, the system took to long to select an ideal algorithm.

Questions

- How exactly does ProfileTM's profiling of memory accesses work?
- The authors modify the STM algorithms to "count readonly and writing transactions separately". Could this have a performance impact?
- How are the oracles created?

Summary A3

This week's paper is about applying machine learning to adaptive transactional memory systems. The key idea is to adapt the used STM algorithms during execution depending on the system environment. There are a lot of different methods to implement the functionality a STM needs. Every approach has different pros and cons and they might excel in different situations.

The presented approach works in 2 stages: the off-line training and the learning during runtime. For the off-line training they use microbenchmarks on every STM algorithm they want to incorporate in their system. A learning tool (ProfileTM) measures the behavior and the throughput of the algorithms and used this information to create an Adaptivity Policy. This policy basically predicts which algorithm for the current environment will be the most effective.

During runtime, they can apply one of two machine learning techniques. The first one is called Case-Based Reasoning. This is a very intuitive approach: the system learns from past experiences and tries to apply those to new problems. For that, it searches for entries with the same or a similar amount of threads as the current workload and from those selects the one that is most similar to the given situation.

The other ML technique is called NEAT Classification. NEAT stands for "Neuro Evolution of Augmenting Topologies". This technique is way more technical: it groups the training data as *k* tuples, where the first entry is the output and all further entries are inputs which produce those inputs. NEAT then computes a function which produces the output for every input. NEAT excels in cases where the program behavior is mathematically related to the choice of the best algorithm.

Based on the off-line training the authors created an "oracle" for each benchmark used in the evaluation. The evaluation shows that both of their approaches are within a 10-7% range of the best-case oracle. But they say that this is not yet satisfactory, as they can theoretically outperform the oracle by exploiting the dynamic phases a program normally has. This will be future work.

Open questions:

1. What is non-transactional work inside a transaction?
2. Why is training with common-date for the target application unfair?
3. What are those 8 features they mention in section 4.3.Similarity and what are they needed for?
4. What are ELA semantics?
5. Why are 1, 2, 4 and 8 threads the only number of threads that will not cause preemption? Why would any other amount of threads cause preemption?

Towards Applying Machine Learning to Adaptive Transactional Memory

1 Summary

The paper proposes a technique to adaptivly select between multiple STM implementations by applying algorithms used in machine learning.

They utilize offline profiling information to select between different STM implementations. To collect this information they use a custom STM system called ProfileTM which will execute only one transaction at once but will still execute transactions on different threads. Using ProfileTM they will execute certain fixed micro-benchmarks on each STM system and gather statistics about transaction throughput and certain factors of possible STM implementation details. This information will then be used as input for machine learning algorithms (in their example class based reasoning and neuronal networks). The ML algorithms will then output an adaptivity policy that will select an algorithm based on application requirements and profiling information at runtime.

During program execution the adaptivity policy will select an initial STM system. While running the system will wait for certain triggers that could be evidence of an unfavorable selection (consecutive aborts or long stalling). When being triggered the system will collect profiling information executing ProfileTM again for a certain number of transactions. The collected profiling information can then be used to select a probably better fitting STM system.

2 Questions

- Is their approach easily portable between platforms and does training help select suitable implementations for the current platform?
- Is the initial selection of an STM based on any program specific profiling information or is it the same for any workload with the same requirements (ELA/weak semantics)?