Project

We are Project 2: Bounded-timestamp register locks, Group 1

having a consensus number of infinity means that this object can be used to give a correct implementation of any object for which you can give a sequential specification. Correctness means Linearizability here.

I think, this is what the book stated about the analogue between the Turing machine and objects with consensus number infinity.

do some experiments

do some benchmarks

do not give a detailed explanation of the algorithm. dont give a proof, why it works or why it is deadlock free or whatever.

benchmarks of the projects should be run on the amd 64 core systems.

you dont need to implement everything that is presented in the paper. filter out the main idea and implement that.

you spawn some threads

let the threads share some objects

one of the performance figures is the running time

OMP\_NUM\_TREADS can control number of threads from outside the parallel region

no Q&A

no project status representation

# General Things

Benchmarks

latency - time per operation (putting something on a queue, popping something off a stack...)

throughput - number of operations per time unit (should increase with number of threads)

fairness - use some counters: how often did a thread acquire a lock etc. Be sure to use thread local counter variables. performance counters in shared arrays will have heavy impact on the performance. ("false sharing" - when operations that are logically not connected cause cache traffic).

all as functions of the number of threads

compare to **baseline** - either another implementation of the concurrent object or a sequential implementation

Do **statistics**: repeat experiment at least 30 times and give averages with confidence intervals. should be enough times such that that the throughput and latency etc. is a stable number.

There will be cases where a certain implementation will work very well and cases, where it is very bad. Try to state these cases.

Test

Do tests, define test cases. Testing does not need to be documented in the report. just do it for yourself

state the properties of the implementation

state properties of algorithms and give worst case bounds (no proofs required). The papers will make some claims: note down these claims clearly.

Memory management

dont bother. its ok to leak

# The Handin

single zip file

[name1]\_[name2]\_amp\_project\_[p.nr.]

Report: 6-10 pages excluding plots and source code

For reproducibility: State the properties of machine, compiler and environment gcc version 4.7.2 (Debian 4.7.2-5)

compile on front end. for benchmark: submit jobs using slurm.

a report with benchmarks and plots and so on

code with makefiles (maybe run it maybe not)

Projects

P1

critical sections of different sizes

this can be a parameter in the benchmark

so do something in the cs

in real memory systems, things don't always happen in the order in which it is written in the algorithm. Maybe you have to take action. Use C++ or C memory model. or use memory fences. to make sure, things happen in the correct order.

it would be interesting to run it on different memory systems. x86 is quite well behaved, so we might not see some of the problems, that we would see on an IBM system.

There will be a lecture about memory models and what you can do in 2-3 Weeks

# Project 2: Bounded-timestamp register locks

Implement Taubenfeld, Lamport, and two out of the other three:

* Szymanski’s solution (Boleslaw K. Szymanski: A simple solution to Lamport’s concurrent programming problem with linear wait. ICS 1988: 621-626)
* Jayanti et al.’s solution (Prasad Jayanti, King Tan, Gregory Friedland, Amir Katz: Bounding Lamport’s Bakery Algorithm. SOFSEM 2001: 261-270)
* Aravind’s solution (Alex A. Aravind: Yet Another Simple Solution for the Concurrent Programming Control Problem. IEEE Trans. Parallel Distrib. Syst. 22(6): 1056-1063, 2011)
* Black-white Bakery (Gadi Taubenfeld: The Black-White Bakery Algorithm and Related Bounded-Space, Adaptive, Local-Spinning and FIFO Algorithms. DISC 2004: 56-70)
* Lamport’s Bakery (lecture version or original)

baseline: Lamport's bakery (unbounded) and Taubenfeld

with performance counters and assertions give a plausibility check of what is claimed in the paper

compare with one of the following locks: pthreads, native C11, simple test-and-set lock, simple test-and-test-and-set lock,

# Additional things he mentioned

Locks lecture slide 12

Make a barrier in your code at the beginning of your benchmark, so that you can be sure, that all threads are actually at the beginning of the benchmark, when it starts.

Locks lecture slide 13

He recommends using openmp.

Here is how to measure time

He has an example of false sharing here: threadtime[] is a shared array with no logical connection between the threads. Can be solved by a second barrier before threads write to it.

Locks lecture slide 15/16

Test-and-set lock

Test-and-test-and-set lock

Variables that can be changed by another thread should be qualified as volatile to prevent program breaking optimizations by the compiler

Don’t use rand() not thread safe (there are side effects… I don’t understand). He uses his own rnd().

S18

He sais: it is always the case, that when a lock crashes, while in the cs, the whole thing is deadlocked. So I guess that is a hint, that we do not need to make threads crash.

S20

Why do we not need an atomic served integer? People are reading it, while I am writing it. If it is only safe, the reader could end up with any integer value.

S23

Avoid false sharing, by padding in a way that two variables do not end up on the same cache line.

S24

use thread\_local to make a variable thread-local (isnt that the same as using private variables in a parallel section?)