

Concurrent Coherent Logic

(coColog System Description)

John Fisher

consultant for ACL project
UiB, Norway, 2005-2009
fisher.r.john@gmail.com

Abstract. This report describes a multi-state model for the design of a search engine for coherent logic using *cooperating concurrent threads*. The proposed model provides *parallel* computations for *disjunctive* cases (split tapes or *branches*) that arise in the operation of an abstract Skolem machine. Initial experiments suggest that this approach provides good *liveness* (time) for coherent theories with splitting rules at modest *space* (memory) costs on current multicore computer systems. We also explain how concurrent processing of branches can help promote the generation of *finite counter models*.

1 Branch logic (one thread)

Suppose that C is a coherent theory specified as a finite ordered sequence of geolog (coherent) rules as described in the theory papers [4],[5], or [6].

This note describes a concurrent multi-state design that parallelizes the sequential algorithm (*QEDF*) described in [7] but now using multiple threads which explore *spawned* branches that arise from the application of splitting rules.

The fundamental unit of parallelization is a geolog tree *branch*. Each such branch corresponds to a Skolem machine *tape* as described in the theory papers. In this design *one branch* is computed within *one thread*. However we will speak of a branch as an executable entity (a Runnable class in the Java implementation).

This first section describes the relevant branch *data* or *state*. When our branch encounters a splitting rule critical branch data must be spawned to new branches which will run in separate threads (Section 2). A *branch manager* monitors spawned branches and reports regarding overall progress (Section 3).

We present the basic concepts by means of a *telling* example. Let us suppose that the current branch (or Skolem machine tape) appears as in (1).

$$\begin{array}{l}
\begin{array}{|c|} \hline 0 \\ \hline \dots \\ \hline 7 \\ \hline \dots \\ \hline 11 \\ \hline \dots \\ \hline f \\ \hline \end{array} \rightarrow \begin{array}{l} true \\ \\ p(a) \\ \\ q(a,b) \\ \\ \end{array} \\
B : \\
>
\end{array} \tag{1}$$

Branches are associated with an *active theory*. The active theory associated with branch B represents the *active rules* which have an binding state on B. Let us suppose that the active theory has an active rule R representing the geolog rule displayed in(2).

$$R.rule : p(X), q(X, Y) => s(X, X) \mid w(X), w(Y). \tag{2}$$

We suppose that R is applicable on B. The intended binding state for R is depicted in 3).

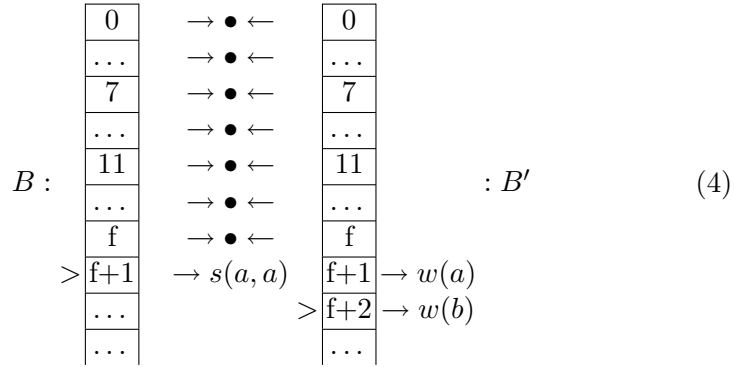
$$\begin{array}{l}
R.state : \begin{array}{|c|c|} \hline X & a \ - \\ \hline Y & - \ b \\ \hline & 7 \ 11 \\ \hline \end{array} \\
\end{array} \tag{3}$$

At this point branch B first spawns a new branch B' which will run in a separate thread and then branch B continues as described in the next section.

2 Spawn a new branch

The cost of spawning new branches includes the branch state container and the state containers of the associated active rules. A significant amount of other data, such as the branch data and the rule forms, are shared via referencing and this means that the copying is quite shallow.

When the branch in (1) is split the results are depicted in (4).



The new branch B' is either supplied by the branch manager from a branch pool or generated anew if the pool is empty.

- 1– the branch data references $0 \dots f$ are copied
- 2– branch B' is extended and its focus realigned
- 3– branch B is extended and its focus realigned
- 4– the active theory associated with branch B' is initialized
- 5– branch B' is handed off to the branch manager to run with focus $f+2$
- 6– branch B continues with focus $f+1$

Initializing the active theory associated with B' (item 4) involves setting the binding state of each *definite* active rule R' for branch B' to mimic the corresponding binding state of active R for state B . This accurately reflects the last binding on the branch before the split and allows the next bindings search to backtrack from a current state rather than starting over at the root of the branch. So, for example, the rule R' (for branch B') corresponding to the rule R used above would have its binding state set to the values of those for R since it is a definite rule (no existential variables in the consequent and no function symbols).

However, for queued rules the binding state would be set back to root because of the requirement for earliest-first-binding explained in [7].

3 Branch management (multi-threads)

The *branch manager* is the main thread and the manager dispatches the initial branch (focus 0, containing only root *true*) and the manager monitors the spawning of new branches.

The branch manager owns a concurrent queue of pooled branches which is modestly initialized to supply new branches. As branches finish

(hit depth bound, find proof, or stuck with countermodel) some of the branches may be requeued to the branch pool. The concurrent branch pool supplies and receives branches in a thread safe manner. The requests for new branches and spawning come from the threaded branches themselves.

When a branch is needed and the branch pool is empty a synchronized method of the branch manager constructs a brand new branch and modifies its state as described in the previous section. These created branches will be placed in the pool when finished, and there is no bound on how large this pool will become, but there is no pressure to enlarge the branch pool if branch threads are finishing and resupply the pool fast enough for requests.

Initially the branch pool has n branches where n is the product of the sizes of the consequents (number of disjuncts) of all the rules.

4 Parallelization metrics

One version of *Amdahl's law* [1] expresses an abstract relationship between speedup benefit B , the proportion of the search subject to parallelization P and the number of processors N .

$$B = \frac{1}{(1 - P) + P/N} \quad (5)$$

Our intent is to find experimental approximations for P based upon trials using experimental values for B and N . We will need to take N to represent the number of virtual cores available, since we are employing multiple threads and not distributing branches over dispersed separate processors. For B we use the ratio

$$B \approx \frac{t_{coColog}}{t_{Colog}} \quad (6)$$

where t_{Colog} is the running time for the sequential version of the program and $t_{coColog}$ is the running time of the multi-threaded version of the program for the same input geolog theory.

Solving 5 for P we get

$$P = \frac{N(B - 1)}{B(N + 1)} \quad (7)$$

The calculated value of P is an estimate regarding what portion of the particular theory can be sped up. Notice that actual *speedup* will have a negative value (because, with speedup, $B < 1$).

There may be relationships between P and metrics measuring the *shape* of geolog trees. Geolog trees tend to have distinctive shape patterns. For example, the graphic in Fig. 1 depicts a geolog proof tree. The figure is a scaled screenshot from a sample run.

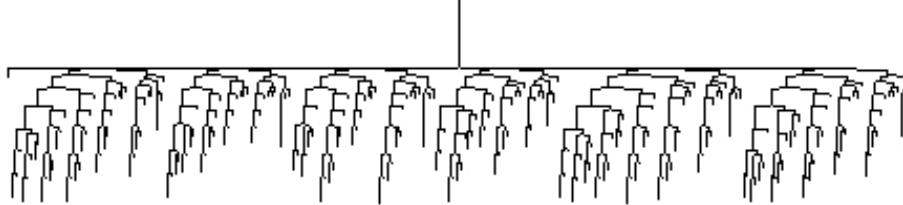


Fig. 1. graphical proof tree *shape* for p1p2

A very simple shape statistic is the *average nodes per branch*:

$$\beta = \frac{\#nodes}{\#branches} \quad (8)$$

For example, $\beta = 3951 \div 231 = 17$ for the tree in Fig. 1 (see results below). This statistic measures, on average, how much work per branch might be done within one thread.

Relationships between P and β would be interesting.

5 Some experimental results

The following subsections report on some initial test results.

5.1 Dual 2.5 Ghz PowerPC G5 Mac

The first set of experiments were run on a two processor system. An attempt was made to achieve runs where extraneous thread demand (other active processes) was minimized. Since this requirement is somewhat tenuous (and often tedious) the timing statistics record *minimum* stable run times over many trials. The Colog program is single threaded. coColog is the multithreaded program described in this note. The inference mechanism for coColog is exactly the same as for Colog, so the timing variance should mostly reflect the multi-threading.

	dpe	mb	nl	p1p2	p2p1	rhp.20	unf	usm
t_{Colog}	9ms	17	67	2585	560	1283	17	12
$t_{coColog}$	11	27	30	1667	332	1023	26	15
$\#nodes$	35	153	100	3951	590	21379	66	31
$\#branches$	3	18	5	231	28	1920	6	1
P	+0.18	+0.27	-1.74	-0.61	-0.84	-0.24	+0.20	+0.15
β	12	9	20	17	21	11	11	31

The larger Theories (p1p2, p2p1, rhp.20) show good time improvements. Theories p1p2 and p2p1 involve the side conditions in Pappus' Theorem in plane geometry. More information is provided in [8] which gives Marc Bezem's original theory sources for p1p2 and p2p1. The input rhp.20 is one of Andrew Polonsky's many early FOL-to-geolog translations of the TPTP problem COM003+3.p (*The halting problem is undecidable*). See reference [9] for a preview regarding Polonsky's translations from FOL to coherent logic. We now have much better translation than rhp.20 but the old version is just fine for testing coColog vs. Colog!

It seems that larger β improves the value of P as expected. Theories mb and unf are small so perhaps the branch pool overhead does not allow speedup. Notice in particular the P metric for the nl theory: Ironically, this says that 174% of the sequential program is subject to speedup! The value of P in Amdahl's law would be ≤ 1.0 of course, but the calculated statistic is not necessarily so restricted. (Our crude statistic obviously needs a refinement.) The usm theory has no splitting rules.

5.2 4 core x86_64 GNU/Linux (kongle.ii.uib.no)

	dpe	mb	nl	p1p2	p2p1	rhp.20	unf	usm
t_{Colog}	2ms	36	59	1049	242	536	39	15
$t_{coColog}$	3	29	42	415	136	353	22	21
$\#nodes$	35	153	100	3951	590	21379	66	31
$\#branches$	3	18	5	231	28	1920	6	1
P	+0.33	+0.24	-0.40	-1.53	-0.78	-0.49	0.77	0.29
β	12	9	20	17	21	11	11	31

6 Future experiments

We continue to experiment with several versions of the coColog prover. A refined version uses something we call *box search* which is well suited to computing finite counter models for coherent theories. For this see

reference [7]. Box search is partially complete both for proofs and finite counter models in the sense that, provided that the box depth (branch depth) and box width (allowed number of branches) forms a large enough box, any proof or finite counter-model can be trapped within the box.

If a coherent logic search tree has multiple branchings then one expects that the branches can distribute their descendant branches into parallel computations. This adds some breadth to the search at the cost of some space but perhaps saving some time. We have shown that coherent logic implemented via Skolem machine computations has a reasonable parallel computation using a multibranching regimen which otherwise adheres to a sequential depth-first approach very closely. Reference [2] describes a parallel SAT checker based upon multibranching and strategies for choosing which clauses to resolve.

More design work may afford a truly distributed approach across distributed processors.

There is another approach to parallelism that is yet to be carefully designed and prototyped: *And parallelism* of rule applications, or multi-threaded rule applications.

References

1. Wikipedia article, *Amdahl's law*, http://en.wikipedia.org/wiki/Amdahl's_law
2. W. Blochinger, C. Sinz, and W. Kuechlin. A universal parallel sat checking kernel. Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2003), 2003.
3. M.A. Bezem and T. Coquand, Automating Coherent Logic. In G. Sutcliffe and A. Voronkov, editors, *Proceedings LPAR-12*, LNCS 3835, pages 246–260, Springer-Verlag, Berlin, 2005.
4. John Fisher and Marc Bezem, Skolem Machines and Geometric Logic. In C.B. Jones, Z. Liu and J. Woodcock, *Proc. ICTAC 2007 The 4th International Colloquium on Theoretical Aspects of Computing*, Macao SAR, China, September 26-28, 2007. Springer LNCS vol. 4711, pp. 201-215.
5. John Fisher and Marc Bezem, Query Completeness of Skolem Machine Computations. In J. Durand-Losé and M. Margenstern, editors, *Proc. Machines, Computations and Universality '07*, Universite d'Orleans - LIFO, Orleans, France September 10-14, 2007. Springer LNCS vol. 4664, pp. 182-192.
6. John Fisher and Marc Bezem, Skolem Machines, *Fundamenta Informaticae*, 91 (1) 2009, pp.79-103.
7. John Fisher, *QEDF Proof Search for Coherent Logic*, technical note, winter 2009, and *Box Search for Coherent Logic*, technical note, fall 2009. See links at <http://JohnRFisher.NET/colog/>
8. John Fisher and Marc Bezem, *Geolog and Skolem Machines*, website discussion of Prolog-based provers and coherent theories. <http://JohnRFisher.NET/geolog/>
9. Andrew Polonsky and Marc Bezem, Proof Objects for Logical Translations, Proc. The 1st Coq Workshop, Munich, Germany 21 August, 2009, pages 49-61 <http://coq.inria.fr/files/coq-workshop-TUM-I0919.pdf>.