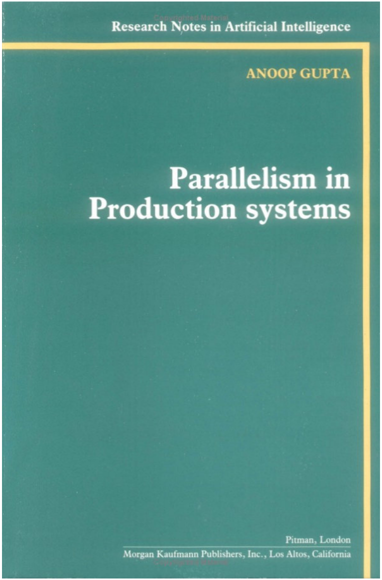
ConcRete Road

Mitchell Keith Bloch

May 9, 2019

A Starting Point



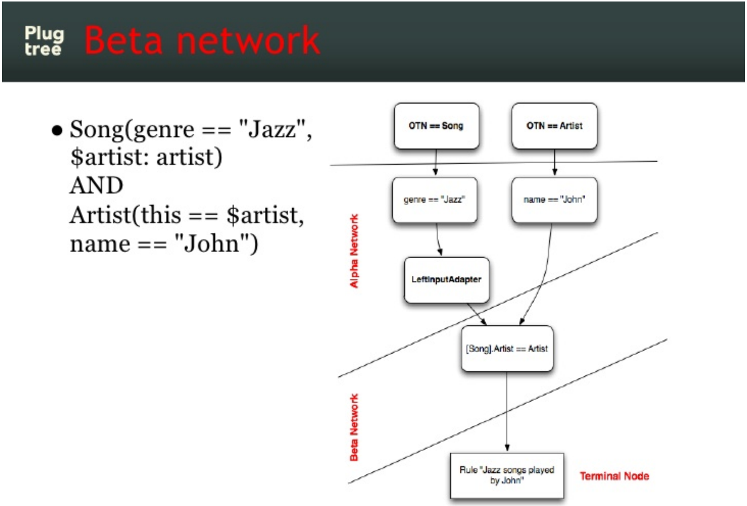
Parallel Production Systems are an Old Idea

- Anoop Gupta's thesis, Parallelism in Production Systems, [Gupta, 1986] provided a good analysis of the problem.
- Production System Machine project's Encore Implementation (PSM-E) was executed on the Encore Multiprocessor. [Gupta *et al.*, 1988]
  - 10-20x speedup for C over Lisp on a uniprocessor
  - 2.8-12.4x speedup using 13 processes
- ParaOps5/CParaOps5 [Kalp *et al.*, 1988] is more widely available.
- UMass Parallel OPS5 exposed parallelism and locking control mechanisms in the RHS. [Neiman, 1992]

Parallel Production Systems are *Contentious*

- Milind Tambe summarized efforts to parallel production systems in Mind Matters: A Tribute to Allen Newell [Tambe, 1996]
- He observed contradictory findings:
  - The field of parallel production systems concluded that increasing numbers of productions will not add to match cost.
  - The field of learning systems concluded just the opposite.
- The Large Systems Effort (LSE) [Doorenbos *et al.*, 1992; Tambe *et al.*, 1992] corroborated their findings:
  - Dispatcher-Soar grew from 2k-12k productions with no increase in match cost.
  - Path-planner-Soar grew from 300-2k productions with a twofold increase in match cost.

What is Rete?



Borrowed from Mauricio Salatino's Drools5 training module

Gupta's Types of Parallelism

1. Production – Completely independent processing of productions
   - Means no sharing of work though
2. Node – LHS nodes can execute in parallel
3. Intra-Node – Multiple threads of execution can work on a single node in the LHS
4. Action – Parallel modification of working memory in the RHS
5. Application – Multiple rules firing in parallel
   - Distinct from #1 in that the RHS executes, not just the matching
   - Possible in Soar due to operator semantics and i-support

Gupta's Types of Parallelism

1. Production – Completely independent processing of productions
   - Means no sharing of work though
2. Node – LHS nodes can execute in parallel
3. Intra-Node – Multiple threads of execution can work on a single node in the LHS
4. Action – Parallel modification of working memory in the RHS
5. Application – Multiple rules firing in parallel
   - Distinct from #1 in that the RHS executes, not just the matching
   - Possible in Soar due to operator semantics and i-support

2-5 are not mutually exclusive!

Locks and Lock-Freedom

- PSM-E, ParaOps5, and UMass Parallel OPS5
  all use locks to prevent race conditions.
- Non-blocking algorithms are… *interesting*
  1. Obstruction-freedom requires partially completed operations to be reversible. All threads can starve without supervision.
  2. Lock-freedom requires a guarantee that at least one thread can make progress. At least one thread must not starve.
  3. Wait-freedom requires a bound on the number of steps for each operation. No thread can starve.

Locks and Lock-Freedom

- PSM-E, ParaOps5, and UMass Parallel OPS5 all use locks to prevent race conditions.
- Non-blocking algorithms are… *interesting*
  1. Obstruction-freedom requires partially completed operations to be reversible. All threads can starve without supervision.
  2. Lock-freedom requires a guarantee that at least one thread can make progress. At least one thread must not starve.
  3. Wait-freedom requires a bound on the number of steps for each operation. No thread can starve.

I asked the question, can a Rete implementation be lock-free?

The answer is *mostly* yes.

Lock-Free Rete

The answer is *mostly* yes.

- Parallel WME insertion ✓ removal ✓
- Parallel rule creation ✓ excision ✓
- Node ✓ Intra-Node ✓ Action ✓ Application ✓
- Propagation stealing ✗

Lock-Free Rete

The answer is *mostly* yes.

- Parallel WME insertion ✓ removal ✓
- Parallel rule creation ✓ excision ✓
- Node ✓ Intra-Node ✓ Action ✓ Application ✓
- Propagation stealing ✗

How does this work? And why the ✗?

What comprises a node in the Rete?

1. [1,2] inputs
   - Technically up to k inputs in filter nodes
     where k is the number of symbols allowed by a disjunctive test
2. A set of tokens from each input,
   where a token is [1,∞] ordered WMEs
3. A decision procedure for emitting output tokens
4. A set of output tokens (implicit from #1-3)
5. [0,∞] outputs

What comprises a node in the Rete?

1. [1,2] inputs
   - Technically up to k inputs in filter nodes
     where k is the number of symbols allowed by a disjunctive test
2. A set of tokens from each input,
   where a token is [1,∞] ordered WMEs
3. A decision procedure for emitting output tokens
4. A set of output tokens (implicit from #1-3)
5. [0,∞] outputs

What's coupled here?

What comprises a node in the Rete?

1. [1,2] inputs
   - Technically up to k inputs in filter nodes
     where k is the number of symbols allowed by a disjunctive test
2. A set of tokens from each input,
   where a token is [1,∞] ordered WMEs
3. A decision procedure for emitting output tokens
4. A set of output tokens (implicit from #1-3)
5. [0,∞] outputs

What's coupled here?

and

What comprises a node in the Rete?

1. [1,2] inputs
   - Technically up to k inputs in filter nodes
     where k is the number of symbols allowed by a disjunctive test
2. A set of tokens from each input,
   where a token is [1,∞] ordered WMEs
3. A decision procedure for emitting output tokens
4. A set of output tokens (implicit from #1-3)
5. [0,∞] outputs

What's coupled here?

#1 & #2 and

What comprises a node in the Rete?

1. [1,2] inputs
   - Technically up to k inputs in filter nodes
     where k is the number of symbols allowed by a disjunctive test
2. A set of tokens from each input,
   where a token is $[1,\infty]$ ordered WMEs
3. A decision procedure for emitting output tokens
4. A set of output tokens (implicit from #1-3)
5. $[0,\infty]$ outputs

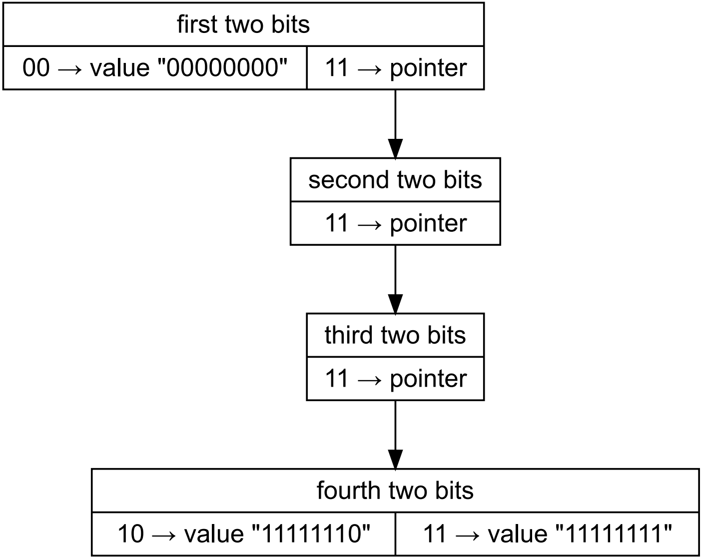
What's coupled here?

#1 & #2 and #4 & #5

Of HAMTs and Ctries

- Lock-free hash tables/maps
- Hash Array Mapped Tries (HAMTs) [Bagwell, 2001] are B-tries where most internal nodes are indexed by a slice of bits from the stored objects' hashes.
  - Insertion/Removal/Modification is done by rewriting to the root and doing an atomic Compare And Swap (CAS).
  - Allows for efficient snapshots: $O(log_W(n))$ storage per snapshot
- Ctries [Prokopec *et al.*, 2012] incorporate some indirection and a significantly more complex GCAS to allow parallel updating of different parts of the HAMT with less contention on the root.
  - Still allows for lock-free snapshots.
  - However, snapshots require more computation and $O(n)$ storage per snapshot.

HAMTs at a Glance



Borrowed from Marek's Introduction to HAMT

Novel HAMT and Ctrie Variants

1. Counting HAMTs
2. Super HAMTs that couple unrelated types
3. Nested HAMTs
4. HAMTs and Super HAMTs nested in Ctries
5. World's first leak-free, non-garbage-collected Ctrie implementation

Novel HAMT and Ctrie Variants

1. Counting HAMTs
2. Super HAMTs that couple unrelated types
3. Nested HAMTs
4. HAMTs and Super HAMTs nested in Ctries
5. World's first leak-free, non-garbage-collected Ctrie implementation

Why care about #2-#4?

Novel HAMT and Ctrie Variants

1. Counting HAMTs
2. Super HAMTs that couple unrelated types
3. Nested HAMTs
4. HAMTs and Super HAMTs nested in Ctries
5. World's first leak-free, non-garbage-collected Ctrie implementation

Why care about #2-#4?

Coupling!

Ctries/HAMTs for Rete

1. Input map
   - Ctrie based on hashes of deciding subset of input
     - Was originally just a big Super HAMT in filter nodes, but Ctries are strictly necessary to reduce contention
   - Containing Super HAMT that counts tokens from the input(s)
     - Duplicate tokens and removals occurring out of order can be ignored
2. Output map
   - Super HAMT of emitted output tokens, connected outputs, and disconnected outputs
     - Connected/Disconnected or left/right unlinking

Design Ramifications

1. 3-variable filter tests are out.
   - The possibility of their existence means WMEs cannot be decoupled using hash values of any of the symbols they contain.
2. Node deduplication requires that one input out of the [1,2] or [1,k] be the first to be attached and the last to be detached.
3. This all works thanks to atomic insertions/removals and corresponding snapshots of our HAMTs.
   - These snapshots are still stored locally.
   - Finding a way to make the iteration over them for message propagation accessible for job stealing is onerous.

Design Ramifications

1. 3-variable filter tests are out.
   - The possibility of their existence means WMEs cannot be decoupled using hash values of any of the symbols they contain.
2. Node deduplication requires that one input out of the [1,2] or [1,k] be the first to be attached and the last to be detached.
3. This all works thanks to atomic insertions/removals and corresponding snapshots of our HAMTs.
   - These snapshots are still stored locally.
   - Finding a way to make the iteration over them for message propagation accessible for job stealing is onerous.

This takes the message passing model for Rete to limit.

Design Ramifications

1. 3-variable filter tests are out.
   - The possibility of their existence means WMEs cannot be decoupled using hash values of any of the symbols they contain.
2. Node deduplication requires that one input out of the [1,2] or [1,k] be the first to be attached and the last to be detached.
3. This all works thanks to atomic insertions/removals and corresponding snapshots of our HAMTs.
   - These snapshots are still stored locally.
   - Finding a way to make the iteration over them for message propagation accessible for job stealing is onerous.

This takes the message passing model for Rete to limit.

https://github.com/bazald/ConcRete

Design Ramifications

1. 3-variable filter tests are out.
   - The possibility of their existence means WMEs cannot be decoupled using hash values of any of the symbols they contain.
2. Node deduplication requires that one input out of the [1,2] or [1,k] be the first to be attached and the last to be detached.
3. This all works thanks to atomic insertions/removals and corresponding snapshots of our HAMTs.
   - These snapshots are still stored locally.
   - Finding a way to make the iteration over them for message propagation accessible for job stealing is onerous.

This takes the message passing model for Rete to limit.

But is it worthwhile?

Design Ramifications

1. 3-variable filter tests are out.
   - The possibility of their existence means WMEs cannot be decoupled using hash values of any of the symbols they contain.
2. Node deduplication requires that one input out of the [1,2] or [1,k] be the first to be attached and the last to be detached.
3. This all works thanks to atomic insertions/removals and corresponding snapshots of our HAMTs.
   - These snapshots are still stored locally.
   - Finding a way to make the iteration over them for message propagation accessible for job stealing is onerous.

This takes the message passing model for Rete to limit.

But is it worthwhile? What if we had gotten it all wrong?

Data-Oriented Design

- Mike Acton sounded the alarm for data-oriented design.
  - This stands in opposition to object-oriented design and to message-passing models.
  - Unity Technologies has taken it to heart with DOTS (formerly ECS).
- At least one large company has explored lock-free algorithms but has decided employees shouldn't use them.

Data-Oriented Design

- Mike Acton sounded the alarm for data-oriented design.
  - This stands in opposition to object-oriented design and to message-passing models.
  - Unity Technologies has taken it to heart with DOTS (formerly ECS).
- At least one large company has explored lock-free algorithms but has decided employees shouldn't use them.

So how does this apply to Rete?

Data-Oriented Design and Rete

- Forget about message-passing and generate big batch jobs?
- Use dumber data structures that are:
  - more cache friendly?
  - easier to do divide and conquer / transform-reduce over?
- Papers claiming Rete-on-GPU exist:
  - [Peters *et al.*, 2013] used OpenCL for ρdf, RDFS, and pD* rule sets.
  - [Guo *et al.*, 2016] used CUDA.
  - Speedup seems modest (9x is as good as it gets for [Peters *et al.*, 2013])
  - Unclear how general purpose these systems were

Data-Oriented Design and Rete

- Forget about message-passing and generate big batch jobs?
- Use dumber data structures that are:
  - more cache friendly?
  - easier to do divide and conquer / transform-reduce over?
- Papers claiming Rete-on-GPU exist:
  - [Peters *et al.*, 2013] used OpenCL for ρdf, RDFS, and pD* rule sets.
  - [Guo *et al.*, 2016] used CUDA.
  - Speedup seems modest (9x is as good as it gets for [Peters *et al.*, 2013])
  - Unclear how general purpose these systems were

I'm working on this now using C++/OpenMP and C#/Unity-Burst.

Nuggets and Coal

Nuggets

1. I wrote the first leak-free, non-garbage-collected Ctrie implementation.
2. I implemented a Rete that is for most intents and purposes lock-free.
3. I made some interesting observations about the design of Rete.
4. Data-oriented design is promising for a faster Rete.

Coal

1. How to benchmark?
2. Lock-freedom is not a silver bullet for avoiding contention in the Rete.
3. Beta-tokens present nasty tradeoffs regarding storage efficiency, cache-locality, mixing of hot and cold data, ...
4. Parallel std::algorithms? OpenMP? TBB? . . . ?

Phil Bagwell.
Ideal hash trees.
ES GRANDS CHAMPS, 1195, 2001.

R. Doorenbos, T. Tambe, and A. Newell.
Learning 10,000 chunks: What's it like out there?
In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 830–836, Menlo Park, CA, March 1992. AAAI Press/The MIT Press.

J. Guo, C. Hwang, and M. Chen.
Using gpu to shorten the match time of rule reasoning based on rete algorithm.
In *2016 International Symposium on Computer, Consumer and Control (IS3C)*, pages 883–886, July 2016.

Anoop Gupta, Milind Tambe, Dirk Kalp, Charles Forgy, and Allen Newell.
Parallel implementation of ops5 on the encore multiprocessor: Results and analysis.

International Journal of Parallel Programming, 17(2):95–124, 1988.

Anoop Gupta.
Parallelism in Production Systems.
PhD thesis, Pittsburgh, PA, USA, 1986.
AAI8702889.

Dirk Kalp, Milind Tambe, Anoop Gupta, Charles Forgy, Allen Newell, Anurag Acharya, Brian Milnes, and Kathy Swedlow.
Parallel ops5 user's manual, 1988.

Daniel E. Neiman.
Design and Control of Parallel Rule-firing Production Systems.
PhD thesis, 1992.
AAI9305874.

Martin Peters, Christopher Brink, Sabine Sachweh, and Albert Zündorf.
Rule-based reasoning on massively parallel hardware.
In *Proceedings of the 9th International Conference on Scalable Semantic Web Knowledge Base Systems - Volume 1046*,

SSWS'13, pages 33–48, Aachen, Germany, Germany, 2013. CEUR-WS.org.

Aleksandar Prokopec, Nathan Grasso Bronson, Phil Bagwell, and Martin Odersky.
Concurrent tries with efficient non-blocking snapshots.
In *Acm Sigplan Notices*, volume 47, pages 151–160. ACM, 2012.

Milind Tambe, Robert B. Doorenbos, and A. Newell.
The match cost of adding a new rule: A clash of views.
1992.

Milind Tambe.
Parallelism matters.
In David Steier and Tom M. Mitchell, editors, *Mind Matters: A Tribute to Allen Newell*, chapter 6, pages 213–217. Psychology Press, New York and London, 1996.