Parallel Gesture Recognition with Soft Real-Time Guarantees

Thierry Renaux Lode Hoste Stefan Marr Wolfgang De Meuter

Software Languages Lab
Vrije Universiteit Brussel, Belgium
{trenaux,lhoste,smarr,wdmeuter}@vub.ac.be

Abstract

Applying imperative programming techniques to process event streams, like those generated by multi-touch devices and 3D cameras, has significant engineering drawbacks. Declarative approaches solve these problems but have not been able to scale on multicore systems while providing guaranteed response times.

We propose PARTE, a parallel scalable complex event processing engine which allows a declarative definition of event patterns and provides soft real-time guarantees for their recognition. It extends the state-saving Rete algorithm and maps the event matching onto a graph of actor nodes. Using a tiered event matching model, PARTE provides upper bounds on the detection latency. Based on the domain-specific constraints, PARTE's design relies on a combination of *1*) lock-free data structures; *2*) safe memory management techniques; and *3*) message passing between Rete nodes. In our benchmarks, we measured scalability up to 8 cores, outperforming highly optimized sequential implementations.

Categories and Subject Descriptors D.1.3 [*Programming Techniques*]: Concurrent programming; D.3.4 [*Programming Techniques*]: Processors; I.5.5 [*Pattern Recognition*]: Implementation

General Terms Algorithms, Design, Performance

Keywords multimodal interaction, gesture recognition, Rete, actors, soft real-time guarantees, nonblocking, complex event processing, multicore

AGERE! 2012, October 21–22, 2012, Tucson, Arizona, USA.

1. Introduction

To improve the quality of interactions between users and computers, interest in multi-touch input, gesture recognition, and speech processing on consumer hardware has recently emerged. To power more natural user interfaces, primitive sensor readings, collected by multiple devices, need to be correlated to create higher-level events.

Hard-coding these complex correlations in imperative programming languages is cumbersome, error-prone, and lacks flexibility [14]. On the other hand, the domain of machine learning requires a lot of training data to build a statistical model of the gesture. Gathering and manually annotating this data (in positive and negative examples) and additionally parameterising important features, is time intensive. Hammond and Davis [12], Scholliers et al. [20], and Hoste et al. [14] demonstrate that declarative definitions for sketch recognition, multi-touch gestures, or multimodal correlation have important benefits on multiple levels. Firstly, they provide important software engineering abstractions to help the programmer to express their intended event patterns. Secondly, they offer an alternative solution compared to ad-hoc implementations when training data is lacking or hard to gather. Finally, expert programmers are able to refine their event correlations with explicit programming code. These declarative approaches all require an inference engine, which compares sensor events with declarative rules describing the gestures.

The Rete algorithm [7] is one possible foundation for such inference engine. It is a forward-chaining, state-saving algorithm that is used to build rule-based expert systems. More concretely, declarative gesture approaches benefit from it as the execution engine incrementally interprets the events of various input sources based on predefined patterns, i. e., rules defining the possible interactions of a human with a computer. Since the majority of the information is constant, the Rete algorithm minimizes the necessary computation that has to be performed whenever a new event takes place, and a corresponding fact has to be is asserted into the knowledge base. As such, it reduces the computational overhead of continuous pattern matching.

In a multimodal system with many possible interactions, the required computational power outgrows easily what to-

day's processors provide in terms of sequential performance. This is problematic for real-time server-sided pattern recognition, for instance to process surveillance camera-input, as well as for embedded devices and mobile phones with various sensors such as an accelerometer, gyroscope, multi-touch, proximity-sensor, and a microphone. A wide range of applications has been proposed to utilize such sensors by extracting meaningful information from the raw data, commonly called gesture recognition. Examples include discovering a when a phone is dropped, detecting whether the user is "throwing" data to another device[1], and performing multi-touch gestures to quickly access information[2]. Additionally, in certain multimodal applications, this information must be correlated to speech-based input. Fusing these primitive and higher-order events easily becomes excessive for a single processing unit, such that utilizing the steadily rising degree of available parallelism becomes a necessity to provide the required degree of real-time interactivity to the users.

We present here a variation of the Rete algorithm called PARTE, built on a graph represented by a set of actors, that provides both scalability and responsiveness. The contributions of our work are:

Design and implementation of PARTE, a parallel Rete engine tailored towards recognition of user interaction patterns with soft real-time [3] guarantees.

Validation of PARTE's real-time guarantees by characterizing the execution properties of the implemented algorithm, ensuring freedom of unbounded loops and freedom of blocking concurrent interactions.

Validation of PARTE's practicality by showing the scalability of the parallel implementation and demonstrating that the sequential overhead compared to CLIPS[4], a highly optimized sequential implementation, can be overcome in the parallel case.

The remainder of this paper is structured as follows: first, in section 2 we will provide a more detailed discussion of the context of multimodal input systems, their requirements and constraints, and the assumptions we can make. Then, in section 3, we will describe our solution, PARTE, in detail and discuss the parallel Rete algorithm used. Afterwards, we will evaluate the resulting system in section 4, characterizing its execution semantics both with respect to non-blocking behavior and with respect to unbounded loops, as well as presenting the performance evaluation. Finally, we will contrast our approach with the related work in section 5 and summarize our conclusions and future work in section 6.

[1] Hoccer, exchanging data using gestures.
Youtube: `http://www.youtube.com/watch?v=eqv8Q6M1O6Y`

[2] Gesture Search for Android
`http://www.google.com/mobile/gesture-search/`

[3] In *soft* real-time systems, the usefulness of results degrades past their deadline, while in *hard* real-time systems the usefulness drops to zero on a missed deadline. Hence, delays in a soft real-time system undermine the system's quality of service, where delays in hard real-time systems undermine the system's correctness

[4] *CLIPS: A Tool for Building Expert Systems*, Gary Riley, 13 March 2011
`http://clipsrules.sourceforge.net/`

2. Context and Requirements

The domain of gesture recognition comes with a set of properties that is different from many domains in which Rete-like inference engines are commonly used. Since we utilize these particularities of the problem domain in the design of PARTE, we will sketch the application domain briefly and detail a list of requirements for inference engines in this domain.

2.1 Inference Engines for Gesture Recognition

To provide a high-quality user experience, an inference engine used for gesture recognition has to correlate events in a timely manner: when a user for instance interacts with a system through a multi-touch interface, changes should be reflected immediately and with a predictable delay to give the user a natural feedback. The same is true for a broader multimodal interaction: when a user gives a series of voice and gesture commands, the right action should be performed without random delays that confuse the user about whether the command has been accepted or not.

Multimodal systems such as Mudra [14] embed inference engines which only tap the computational power of a single processing unit. However, the rise in sequential processing power offered by single processing units is stagnating, because efforts to increase clock-speed, instruction-pipeline depth, memory-bus width, and cache size, offer diminishing returns. This severely limits the possible number of patterns, their complexity, and the rate of events the system can handle. The only way to recognize more complex user interaction patterns without undermining the user experience by increased delays, is to embrace parallel processing power.

In addition to recognizing patterns in a timely manner (i. e., low latency), the system also needs to guarantee predictable response times (i. e., predictable, real-time latency). This ensures that the system always feels interactive and responsive. Akscyn et al. [2] show that long delays in interactive systems can distract users, and even cause them to stop using the system altogether. Consider for instance a user of a multi-touch gesture recognition system, who taps a certain location. If the user interface does not reflect this change within the timeframe users have grown to expect, they will assume the command was not received, and may tap again. When the system then finishes processing the overdue gestures, the action will be executed twice. Users will rightfully blame the gesture recognition system for this mistake. To prevent such errors, the detection of complex user interaction patterns should happen within a timeframe that can be predicted reliably up front.

However, the requirements of responsiveness and predictable runtime conflict: to offer the best performance on

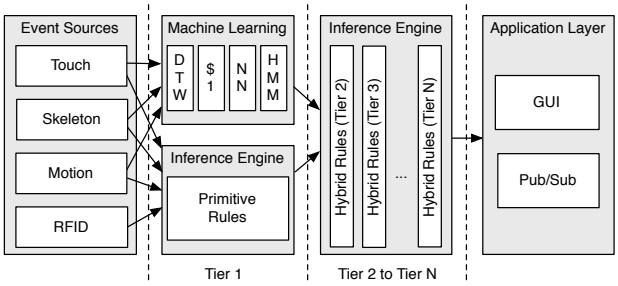

Figure 1. Contextual Framework

current hardware, the rule engine needs to use the available parallelism, while to ensure responsiveness it should provide soft real-time guarantees. Existing rule engines do not combine both requirements. They either are single-threaded in nature, or do not guarantee predictable worst-case execution times.

To give an example, Figure 1 visualizes the data-flow of the multimodal approach presented by Hoste et al. [14]. Event sources such as multi-touch displays, skeletal tracking [21], accelerometer readings or the history of RFID tags contain potential valuable patterns that need to be processed. This unified architecture allows for low-level events to be processed using machine learning-based approaches, as well as declarative definitions. Fusion and refinement of resulting higher-level events can be handled by consecutive declarative rules.

Given these observations, we will use a tiered architecture for event processing. In this architecture, rules of $tier\ N$ can consume only events that were generated by lower-level tiers (1 to $N-1$). It enables developers to easily modularize and compose their rules. For instance, a *wave* gesture can be composed of two lower-level gestures *flick right* and *flick left*, which themselves where extracted from the low-level skeletal data provided by a Kinect[5] system. A declarative approach enables this kind of efficient composition and helps developers to improve gesture recognition code. Enforcing tiering however, implies that a rule of $tier\ N$ should not insert additional lower-level events to help pattern classification on lower levels. Although certain multimodal use-cases benefit from using high-level event information to improve the accuracy of lower-level event detection [14], avoiding feedback loops is required for computational predictability.

Finally, the application layer uses a publish-subscribe model to register for high-level events processed by the inference engine. Depending on the application, it is useful to support different subscription modi. Topic-based subscriptions are used to filter by the type of the event and are the most common ones. However, for instance GUI components use content-based subscriptions to only react on events that happen at a specific spatial location. To enable such application-specific usage, the system needs to provide the necessary extensibility.

This contextual framework is tailored to the domain of multimodal interaction and gesture recognition and guided the design of PARTE. However, similar properties can be found in the broader context of *Time Series Analysis* and *Complex Event Processing*, including domains such as algorithmic stock trading and monitoring security breaches.

2.2 Requirements and Assumptions

By restricting the generality of the Rete algorithm and tailoring it to our application domain, we can make design decisions that simplify the implementation and enable us to achieve the desired properties.

The main target for the system will be commodity multicore hardware. Thus, we will assume shared memory between cores and the presence of a cache hierarchy with memory coherency guarantees.

An important requirement to achieve bounded execution time is that rules are constructed without feedback loops. Based on the practice of tiering, which we outlined in subsection 2.1, we will disallow direct feedback loops and assume that the results always represent higher-level events, i. e., events from a higher tier in the system. Those events may then be fed ("asserted") back into the same inference engine, but will activate a disjoint subgraph of the Rete network: when the rule causing the assertion was on tier n, all rules which cause the event were on tiers equal to or lower than tier n, whereas every rule activated by the newly asserted fact is on tiers strictly greater than n.

Since the ordering of events is an application specific issue, it needs to be handled explicitly as part of the rules. A higher-level event might require the timestamp of the first low-level event in a sequence, the last one, or the time span in which all the related lower-level events occurred. The choice of this timestamp or time span depends on the semantics of the declarative rule, so this choice cannot be automated. Such information therefore needs to be constructed and provided to the next tier explicitly, if temporal order between higher-level events needs to be known.

Related to this assumption is our interpretation of the semantics of events as being permanent. Thus, for the intended use case, it is not necessary to enable retraction of facts, i. e., events will not be removed from the system as part of the action of a rule. Instead, we assume that a higher level rule can always subsume events if necessary. This enables us to avoid the need for *conflict resolution*: conflict resolution is commonly used in rule-based systems to order the execution of rule activations, and enable retraction of facts and subsumption of rule activations. For the intended use case, however, it is desirable that all rules will always be triggered and subsumption is deferred to a higher-level tier. The ordering

[5] Kinect, Microsoft's motion sensing input device
`http://www.xbox.com/kinect/`

hence does not determine the result, and conflict resolution serves no purpose.

The semantically indefinite validity of events entails that the data structures representing events are not removed from working memory by the rules themselves, hence must be removed automatically by the system to prevent the working memory from growing unboundedly. For this, a sliding window of events which are relevant to the current reasoning process can be used. We will require events to be correlated by timestamp, so that the temporal dependencies between events can be used to statically compute their maximum useful lifespan: at any point in time, only those events can be part of a new pattern, for which there exist rules correlating them with other events that occurred within the lifespan of the first event.

Classic rule-based engines are employed in business environments in long-running systems that need to be adaptable and allow changes to the rule set at runtime to avoid downtime. However, this leads to additional complexity and is not required for the given scenario. Thus, we assume that in games and user interface applications only static sets of rules are used and that it is sufficient to determine the set of rules at startup time.

A final requirement is that all event sources have an upper bound on the rate with which they emit events. This upper bound is necessary to enable an estimation of the maximal load of the system.

Summarized, the important assumptions are:

- Rules are free of feedback cycles and produce results for a higher-level tier only.
- Activations of different rules do not require ordering.
- Temporal dependencies are solved in an application specific way by rules.
- Events never need to be retracted from the system. Event subsumption is done on a higher tier. Preventing memory leaks is handled by making events expire when they are no longer useful for the reasoning process.
- The set of rules is known and fixed at startup time.
- All event sources have an upper bound on the rate with which they emit events.

Based on these assumptions and the previously given context, the requirements for a parallel gesture recognition engine are the following:

Soft Real-Time Guarantees The detection of user-interaction patterns has to complete in a predictable amount of time to give the user appropriate feedback.

Efficiency Beside providing predictability, the rule processing has to achieve sufficient efficiency to satisfy constraints on the response time required for interaction with humans. Miller [17] identified three threshold levels in human attention, based on the order of magnitude of seconds that one has to wait. Response times in the order of tenths of seconds are perceived as instantaneous and response times of around one second are perceived as a fluent interaction. For a system detecting user-interaction patterns, the interaction should at the very least be fluent, and preferably instantaneous, i. e., in the sub-second range.

Scalability on Multicore Hardware The performance of the system needs to improve with an increase in the number of available processing cores, relying on a shared-memory architecture.

Optimized for Continuous Event Streams The production system has to be tailored for complex event processing on event streams with a bounded event rate. The event streams are assumed to be infinite and processing has to be online (in contrast to off-line batch processing systems).

Extensibility and Embeddability The system needs to support user-defined functions to process and correlate events, and to produce results on rule activation to be extensible. Domain specific tests are required to facilitate for instance testing of spatial properties of coordinates. For embedding into existing systems, it is necessary to produce the expected result format by invoking callbacks or sending messages to the consuming tier.

3. PARTE

PARTE is a production system using a variant of the Rete algorithm to detect user-interaction patterns. To that end, it transforms a set of declarative *if-then* rules into a directed acyclic graph, and uses this graph to match facts. PARTE is designed to be scalable on parallel systems, as well as to satisfy the requirements and assumptions described in subsection 2.2.

This section first describes how the solution is embedded into the context of gesture recognition, and how it interacts with the main components in such an environment. Then, the architecture of PARTE is described and a high-level overview over the solution strategy is given. Finally, we detail the solution and discuss implementation decisions that are essential to satisfy the posed requirements.

3.1 Architecture and Embedding into Gesture Recognition Context

As outlined in subsection 2.1, inference engines such as PARTE are mediating between the raw input devices and high-level consumers such as application logic. For engineering reasons, such systems use tiered architectures to gradually enrich the semantics of the events. PARTE can be applied at multiple tiers in such an architecture. In such a scenario, PARTE would process incoming lower-level events from the a set of event sources, based on a given set of rules which describe relevant patterns that need to be recognized

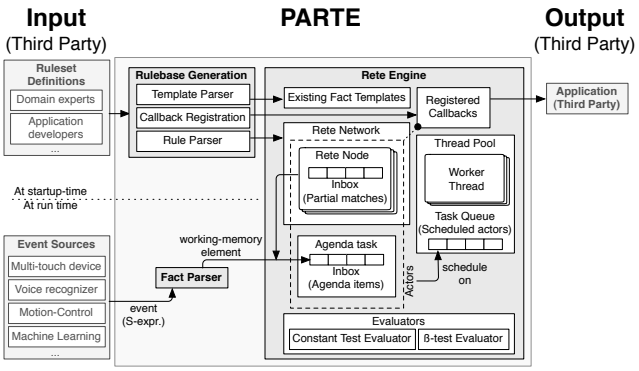

Figure 2. The architecture of PARTE

```
(defrule detectZShape
   ?hA <- (horizontal-drag)
   ?hB <- (horizontal-drag)
   ?diagonal <- (down-left-drag)
   (test (endMeetsStart ?hA ?diagonal))
   (test (endMeetsStart ?diagonal ?hB))
   (test (chronologically
              ?hA ?diagonal ?hB))
=> (reportZShapeCenteredOn
      (avg ?hA.startX ?hA.endX
           ?hB.startX ?hB.endX)
      (avg ?hA.y ?hB.y)))
```

Listing 1. A possible rule for gesture recognition

in these event streams. Thus, PARTE is part of the middleware for building such applications.

Figure 2 depicts the architecture of PARTE with potential input sources on the left, and potential consumers at the right hand side. Rules, the templates describing the facts' structure, and the facts themselves are to be encoded as S-expressions and get converted to the internal representation by a set of parsers. A pool of threads is maintained by the engine, as well as a task queue on which the actors are scheduled. Finally, reentrant evaluator functions are provided to evaluate the test-expressions specified by the rules.

The Rete network itself is constructed from the set of rules inserted into PARTE at startup time. The rule parser converts the rules to an abstract syntax tree (AST), validating their semantics while doing so. From the AST, it then builds the directed acyclic graph as prescribed by the Rete algorithm [7]. When multiple nodes are required which select for the same type of event, the parser reuses the first node selecting for that type it created. More involved node-reuse and optimization strategies are not yet implemented. After computing the Rete graph, PARTE computes the indices ("lexical addresses") of slots within facts and of facts within partial matches. As such, once the system is running, lookup of attributes can be replaced with a constant-time indexed memory access. Finally, PARTE creates a set of actors and links them up to each other to constitute the Rete network.

Apart from the nodes in the Rete network themselves, the Rete algorithm requires another data structure: the agenda. This agenda reifies the FIFO queue of I/O actions to and from PARTE that have to be performed. The only form of input which PARTE accepts at runtime comes in the form of the assertion of new facts, for which *assert* agenda items are used. When the agenda task processes an *assert* agenda item, the event specified in that item is propagated to the inboxes of the Rete network's entry nodes. Inversely, to communicate results to the outside world, PARTE supports user-defined functions, which result in the scheduling of *user function* agenda items on the agenda. When such an item is processed, the corresponding callback function is called.

Since user-defined functions are plain C functions, they *can* technically perform whatever I/O or other time-consuming and/or blocking operation they want, but *are presumed* not to do so. If a rule should require something which is not normally considered a good match to event-processing, such as reading from a file, the user-defined function should dispatch the job of reading the file to a worker thread provided by the application hosting the PARTE engine, in a non-blocking way. That thread can then read the file and assert an event into the systems with the contents of the file.

In addition to the *assert* and *user function* agenda items, PARTE currently supports *print* and *terminate* agenda items which respectively print a string to the console and halt the engine, and could technically have been implemented in terms of *user function* agenda items. PARTE does not support *retract* agenda items, since in our event processing context, facts get removed from the fact base automatically when they expire.

Items on the agenda are processed sequentially, but in parallel with the processing of network nodes. Because of its interaction model, the agenda is represented by an actor as well.

The S-expression in Listing 1 gives an example of a high-level motion gesture rule that can be processed by PARTE. The expression defines the rule `detectZShape`, which describes how two horizontal drags and a down-left drag can combine into a Z-shape. Line by line, the rule binds two events of type `horizontal-drag` to the variables `?hA` and `?hB`, and binds an event of type `down-left-drag` to the variable `?diagonal`. Then, it uses the user-defined functions `endMeetsStart` and `chronologically` to verify that the shapes follow each other both spatially and temporally. As a consequent to the recognition of the Z-shape, the rule specifies that the callback `reportZShapeCenteredOn` should be called, passing the average `x` and `y` slots of the shape's points. Figure 3 shows a Rete graph and the flow of the facts along the edges when the facts at the lower-left corner are

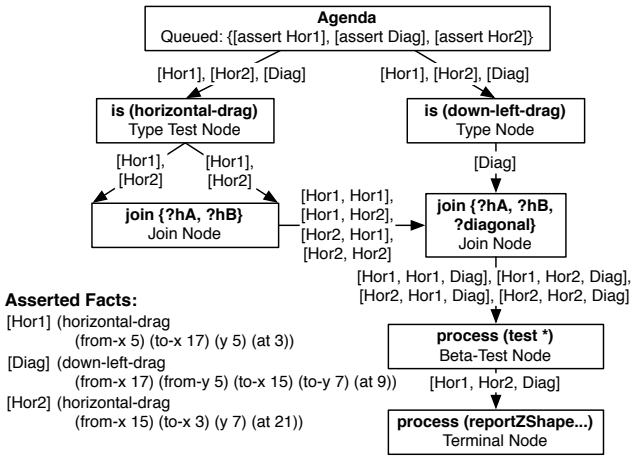

Figure 3. Flow of data through the Rete network specified by the rule in Listing 1

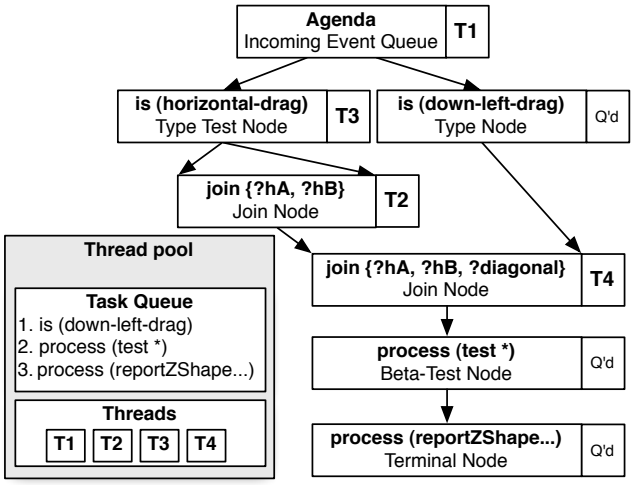

Figure 4. A potential runtime snapshot of a PARTE system detecting the pattern specified in Listing 1

asserted into the Rete network. Lists delimited by square brackets denote tokens, the units of communication between the nodes in the network.

3.2 Parallel Execution Model

For the description of the execution model, we will use the metaphor of the actor model [1] to map each node of the Rete network to its own actor, executing independently. While such an approach does not enable us to utilize potential data parallelism for the matching inside a node, it enables a high degree of parallelism throughout the network. Even in situations where only parts of an actor network are used frequently, this approach enables pipeline parallelism, enabling scaling on multicore processors.

Furthermore, the directed acyclic graph (DAG) structure of a Rete network, and its structural properties in terms of edges between nodes provide a ideal foundation to apply non-blocking data structures to gain predictable upper bounds for execution time. We utilize these characteristics to provide the desired real-time properties.

Execution Model As indicated above, the individual actor nodes of the Rete network are the parallel units of computation. The DAG of the Rete network thereby forms a task-dependency-graph for the match phase of the fact processing. This means that every actor node needs only wait for information from their predecessors in the Rete graph, and only send data to their successors in the Rete graph. This entails that the same spatial and temporal efficiency that the Rete algorithm offers for matching facts to rules, also ensures low contention for shared resources: every node's communication channel is contended for by at most two predecessors and its own thread of control.

The Rete algorithm's approach of passing tokens between nodes makes it map very well on message-passing between actors. Especially, since the step of processing an incoming token can be seen as an atomic operation by the rest of the system that does not require the notion of shared memory. In the implementation of PARTE, the nodes of the Rete network have an inbox, which is realized with a nonblocking queue, in which predecessors put the incoming tokens. A node dequeues tokens from its inbox one-by-one for processing. Because of the nonblocking nature of the inboxes, we avoid the potential for deadlocks and livelocks.

To prevent starvation, every actor, i. e., every Rete node and the agenda, are scheduled on a thread pool using a round-robin scheduler. Figure 4 shows a possible snapshot of a running PARTE system which contains only the rule specified in Listing 1. Four threads are allocated in the thread pool, meaning four of the seven actors can be active at the same time. The other three actors remain queued on the task-queue.

Non-Blocking Data Structures The only form of inter-thread communication on which PARTE depends comes in the form of messages sent to actors' inboxes. Since some nodes and the agenda have multiple predecessors in the Rete graph, we chose the n-producer/m-consumer FIFO queue design by Harris [13]. The list offers lock-free inserts at the front and deletes at the back, and contention is localized to only the element that is inserted or removed, meaning that in lists with two elements or more, enqueueing and dequeueing can happen simultaneously without interfering with each other. Each node of the list consists of a pointer to the data and a next-pointer. The queue always contains at least one such node, with NULL-ed out data: the 'dummy' node.

To enqueue an element in the non-blocking queue, a new list node is created, and its data pointer filled in. The algorithm then enters a loop in which it tries to enqueue the newly created node. To this effect, it grabs the current tail-pointer of the queue, looks at its next-pointer, and if it is

not NULL, help the other thread which must have been responsible for adding a new node past the tail, by attempting to atomically compare-and-swap the next-pointer to the queue's tail slot. If the tail's next-pointer is NULL, then the algorithm attempts to compare-and-swap its own newly created node to the tail's next-pointer. If that succeeds, the algorithm breaks out of the loop; otherwise it keeps looping. After breaking out of the loop, the algorithm still has to compare-and-swap the newly created node to queue's tail-pointer, but if that fails (i. e., when another thread has overwritten the tail-pointer since), no correcting action has to be performed: the other thread will have set the correct tail-pointer. Because of the construction of the Rete network with its limited number of producers and consumers, as well as the semantics of how the lists are used, i. e., how items are inserted, the retry-loop is bounded and local as well as global progress are guaranteed.

Dequeuing works similarly, however, since all lists have only a single consumer, a simple compare-and-swap on the head-pointer to the head-pointer's next-pointer is sufficient. The case of the empty list is implicitly handled with the dummy node.

Memory Management Since PARTE is implemented in C++, memory management becomes an issue that needs to be handled carefully. In our implementation, all facts, i. e., tokens in the Rete network are handled by value and the consumer is responsible for freeing them when they expire. Expiration of events can be determined from the data local to the actor in which the tokens are stored, and therefore does not require a global synchronization effort as is the case in systems where events are only removed after a request for retraction has percolated through the Rete network. PARTE makes use of the timestamps carried by tokens to determine not only how long those tokens still have to be preserved, but also to implement a logical clock with which nodes can know what the oldest point in time is from which their predecessors still may have unpropagated events. By maintaining the invariant that tokens are always propagated in-order, the node actors can know whether new tokens that still can correlate with stored tokens can be expected, and discard tokens for which this is not the case.

More complex is the situation of managing the list elements for the lock-free list implementation. Since they are inherently subject to race conditions, we use a solution similar to reference counting proposed by Michael [16]. All operations on list elements must maintain a set of *hazard pointers*. The hazard pointers are kept in a contiguous piece of memory, and their number depends on the number of threads as well as the use of the data structure. Each thread is associated with a subset of these pointers for its own use. Whenever an operation on a list takes place, a thread explicitly stores pointers to the elements which are *in use* in its thread's section of the hazard pointer array. When an element is deleted from a list, a memory reclamation operation is triggered, which iterates over the hazard pointers and conservatively does not delete any list element which is referenced by a hazard pointer. Since the hazard pointers are visited in ascending order of index in the hazard pointer array, any interleaving of threads using the non-blocking linked-list and the thread reclaiming the linked-list's elements will encounter *at least* all elements that are *in use* at that moment. To this effect, the list's operations may only shift hazard pointers in increasing order of the index.

Because the number of hazard pointers per thread is a small constant, and the number of threads is constant, the amount of memory that is no longer in use, yet not yet reclaimed, is bounded by a small constant per thread, and a reasonably small constant in total.

4. Evaluation

In subsection 2.2, we outlined the requirements for a parallel inference engine used in the context of parallel gesture recognition. This section will discuss how PARTE satisfies these requirements. First, we will discuss the general requirements of how PARTE is optimized for complex event processing (CEP) and how it is embeddable into existing systems. The second part of the evaluation will evaluate the soft real-time properties of PARTE by arguing that the general algorithmic properties and the boundedness of the evaluation process provide the desired guarantees. Finally, we will discuss the performance aspects by comparing the single threaded performance of PARTE and CLIPS, as well as demonstrating scalability of commodity multicore hardware.

4.1 Extensibility and Embeddability

PARTE is designed to be a middleware mediating between the low level of event sources and the application level. To that effect it is a self-contained system, managing its own memory and interacting with other systems via a simple API and callback methods. Applications using PARTE must provide a ruleset and register user functions and callbacks. Event sources need only inform PARTE of new events. The inference engine is continuously running and processes the incoming events as soon as they arrive, maximizing throughput. Through this low coupling between PARTE and the remainder of the system, PARTE is a reusable, easily embeddable software component. The notion of custom user-defined test functions and actions enables PARTE to process arbitrary events and produce output in whatever format the application requires.

4.2 Continuous Event Streams

Since the input devices are assumed to continuously produce events, PARTE was designed to handle expiration of facts automatically to avoid unboundedly growing fact bases. The sliding window mechanism explained in section 3.2 causes the expiration of events which are no longer relevant. This removes not only the burden of manual memory manage-

ment from the application-level; it also reduces synchronization overhead as *retract* messages need not be sent and processed separately. This results in a speedup compared to a version which would not use automatic expiration, as well as improving the scalability.

4.3 Soft Real-Time

For the assessment of the real-time properties of PARTE, we will rely on the algorithmic properties only. Thus, we will disregard architectural issues such as microprocessor architectures [6] and operating system aspects [15]. Instead, we will give an informal argument to demonstrate that the pattern matching is done in a bounded number of steps, which all complete in a bounded amount of time.

We will therefore demonstrate the soft real-time properties of our system by showing that a predictable bound exists on *a*) the time every *turn* of an actor requires; on *b*) the time required for scheduling actors; on *c*) the time required for passing messages between actors; on *d*) the amount of actors; and on *e*) the amount of turns per actor that are required to detect a pattern.

For this first requirement, more information about the inner workings of the actors is required. We introduced five types of actors in Figure 3: the agenda, type test nodes, join nodes, test nodes, and terminal nodes. In the case of the agenda, type test nodes and terminal nodes, this requirement is trivially met. They all perform a constant amount of work, respectively executing one agenda item, performing a single equality test, and scheduling a number of actions which cannot change at run-time. For test nodes, the situation is barely different: the arithmetic expressions and (in-)equality tests they evaluate are fixed at compile-time, so an upper bound on their runtime can be computed. For the join nodes, the argumentation is a little more involved. The only variable factors in a join node's runtime, however, are the number of variables to unify and the number of fields that have to be bound to those variables. Both are explicitly specified in the ruleset, and therefore known at the time the Rete graph is being compiled. Thus, an upper bound on join nodes' runtimes can be computed.

The next two requirements are closely related. Both the task-queue and the actors' inboxes are implemented as non-blocking data structures. Since the structure of the Rete algorithm limits contention on the actors' inboxes, and Rete nodes cannot generate an arbitrary amount of tokens before having to wait for new incoming tokens, an upper bound on the time required to enqueue and dequeue exists.

The requirement for an upper bound on the number of actors is trivially met, as all actors are statically allocated at startup time.

The last requirement is fulfilled thanks to tiering. By definition, the Rete DAG is acyclic, and by enforcing tiering, we prevent the possibility to make cyclic structures via the feedback loop constituted by the agenda. As such, not only the width but also the depth of the loop-unrolled graph is bounded and known for a given ruleset. Since we require a known upper bound on the rate at which events can be asserted into the system, and communication happens via FIFO queues, the maximum number of turns required before all events currently in the system are processed can be computed.

By showing that an upper bound can be computed on the amount of time PARTE requires to detect gestures, we have demonstrated that PARTE offers soft real-time guarantees.

4.4 Performance Evaluation

To evaluate the performance of PARTE, we follow the methodology proposed by Georges et al. [8]. The benchmarks were executed on a Mac OS X 10.6.8 workstation with two Xeon E5520 processors at 2.26 GHz. Neither Turbo-Boost nor hyperthreading could be disabled. Thus, Turbo-Boost can lead to up to 12% higher sequential than parallel performance. However, the measurements of sequential performance remain directly comparable. Both CLIPS and PARTE were compiled with maximum optimizations (-O3) using the GCC 4.2.1 compiler shipped with OS X.

Every benchmarked configuration is run at least 30 times, and is automatically run additionally until a confidence level of 95% is reached. The benchmark results in Figures 5 and 6 are visualized with beanplots to show the distribution of measurements instead of simple overgeneralizing averages.

We used 13 different benchmarks for the evaluation. Each benchmark consists of a set of rules and a set of pre-generated events to be fed into the system. The benchmarks include microbenchmarks to measure the performance of variable binding, different fact sizes, unification, and β-tests. Furthermore, we used a number of kernel benchmarks designed after common gesture rules to assess the performance of rules with complex tests, and tests that use computational intensive user functions. For motion detection such tests are typically trying to find the spacial relations of a group of points and movements.

We will first look into the efficiency of our system by comparing the runtime performance of PARTE running on a single thread with the runtime performance of CLIPS. CLIPS is an open-source and highly tuned sequential implementation of the Rete algorithm and forms the basis of multiple other production systems, such as PRAIS [9] and Lana [3]. After our evaluation of the efficiency, we will look into PARTE's scalability by investigating the effect of increasing the number of threads allocated for the system.

Efficiency To assess the sequential efficiency of our implementation, we compare PARTE to CLIPS. Our goal is to demonstrate that PARTE, in its current unoptimized state has acceptable sequential performance characteristics in direct comparison. Thus, the performance in kernel benchmarks based on the gesture recognition use case as well as computational intensive workload should be in the same order of magnitude.

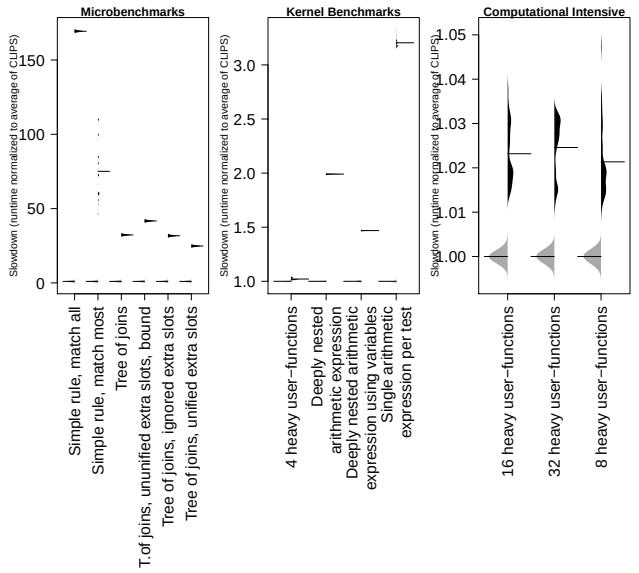

Figure 5. Beanplot comparing CLIPS to PARTE in single threaded execution. Microbenchmarks show architectural overhead. Kernel benchmarks representing typical gesture rules show competitive performance. Computational intensive rules, typical for motion detection rules, show less than 5% overhead.

Figure 5 depicts the results in form of asymmetric beanplots. For each benchmark, the results have been normalized to the average of the CLIPS results. The distribution of the CLIPS results are depicted in gray, while the results for PARTE are shown in black.

The first graph shows the results for the microbenchmarks. They demonstrate the overhead of message-passing and the performance cost compared to CLIPS. This overhead comes partially from the *by-value* semantics used for the messages and partially from the lock-free queues. Both still have optimization potential, but such pathologic microbenchmarks will always point out the higher overhead compared to a direct sequential implementation as employed by CLIPS.

The second graph shows the results for kernel benchmarks with characteristics found in the gesture recognition rules used by Hoste et al. [14]. Here we see that PARTE delivers comparable single threaded performance on the same order of magnitude as CLIPS, without its highly tuned implementation.

The third graph shows results of computational intensive rules, as they are found in complex motion recognition rules that need to correlate spatial coordinates of many events. These kind of rules are the most relevant for advanced uses cases based on input devices such as massive multitouch or 3D cameras. Here, the evaluation of the test expressions is the most intensive part, and PARTE has comparable performance to CLIPS.

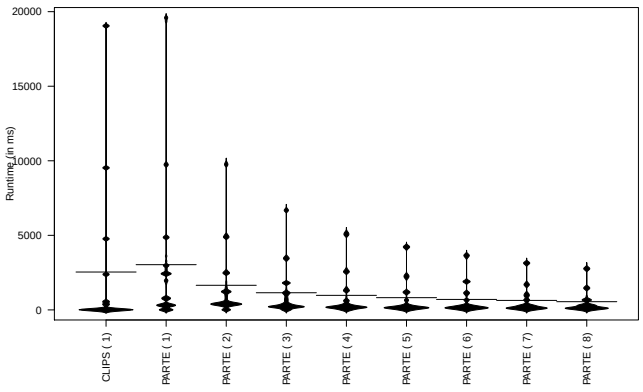

Figure 6. Beanplot showing distribution of benchmark results for 1 to 8 worker threads.

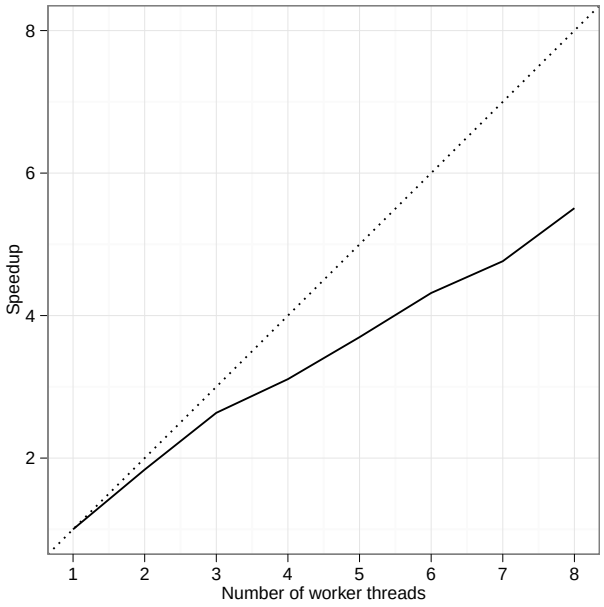

Figure 7. Speedup graph, showing PARTE's scalability with number of worker threads compare to ideal speedup (dotted line).

Scalability A beanplot of the combined results of all benchmarks is depicted in Figure 6. This graph gives an impression of the distribution of all benchmark results for a varying number of worker threads. The little accumulation points along the runtime axis indicate the increased number of measurements at that point, which coincide with specific benchmarks. The horizontal bar indicates the average.

Figure 7 depicts results as a speedup graph to emphasize the scalability of the system. The dotted line indicates the ideal speedup—a speedup which increases linearly with the amount of processors. The graphs show that PARTE scales well in the average case. The system's design offers the

significant advantage of spreading the load over multiple threads. When that load gets heavy, PARTE can effectively benefit from the processing power of different processing units. This form of pipeline parallelism is ideal for rules that rely on complex tests such as used in gesture recognition. For the intended use case, the parallel decomposition created by PARTE enables the system to benefit from a close to ideal scaling up to 8 worker threads.

4.5 Discussion

Our choice of actors as the unit of parallelization stems from the strong similarity between the passing of tokens between nodes in the Rete algorithm on the one hand, and message passing between actors on the other hand. The actor model provides a nice metaphor for the construction of a graph of interlinked nodes which share data only by explicitly passing it to their successors in the DAG.

Some limitations exist in the current version of PARTE. Since test-expressions are separated from the nodes that join two branches, negation-as-failure is not supported in PARTE. Because of the parallel execution model, the implementation of negation requires additional communication and does not fit into the current model. However, gesture recognition systems can work without a notion of negation, but the addition of it would be worthwhile for other use cases.

Furthermore, the coarseness of parallelization can further be tuned, as in some situations the actor-based approach does not expose all options for parallelism. Computational intensive test functions such as required for gesture recognition could benefit from parallelizing the match on facts inside a node/actor. Inversely, in other situations the actor-based implementation using message-passing and lock-free queues imposes a high overhead which could be reduced by merging nodes appropriately.

Another area that could be improved is PARTE's scheduler for the actors, which focusses on correctness and real-time properties. Currently, actors are scheduled although their inbox is empty. Improved scheduling which preserves soft real-time guarantees is planned for future work.

5. Related Work

The Rete algorithm is widely used, and has had many adaptations to both fit real-time requirements and to fit parallel processing.

5.1 Parallel Rete

Gupta et al. [11] measure that the matching is the most computationally expensive task and takes up to 90% of the execution time in production systems. Consequently, most effort has been dedicated to parallelizing the match-phase.

One of the best known Rete derivates focussing on parallelism is the TREAT algorithm by Miranker [18]. In TREAT, for every condition element, the matching facts are stored. This makes TREAT a state-saving algorithm, but less so than Rete, which in addition to those *alpha memories* stores the matching sets of facts for combinations of condition elements that appear in the rules, in *beta memories*. At the other end of the spectrum is the production system proposed by Oflazer [19], which stores the matching sets of facts for *every* combination of condition elements, regardless of whether they appear in the rules. Both TREAT and Oflazer's machine diverged from the traditional Rete algorithm to reduce the need for synchronization, thereby opening options for parallelism. Both approaches had the foreseeable drawbacks: TREAT spends a lot of time recomputing matches for entire patterns, and the combinatorial explosion made Oflazer's machine consume a lot of working memory, in addition to spending a large amount of time computing combinations of facts which may never get used. PARTE, which sticks to the traditional Rete algorithm has none of these drawbacks. It does not decouple the different threads of execution by performing too little or too much work to be able to skip synchronizing, but instead focusses on reducing the overhead of the synchronization. In addition, it uses automatic expiration of facts, which halves the number of inter-node communication that has to take place compared to Miranker's and Oflazer's system with manual retraction.

A different approach is taken by Aref and Tayyib [3], whose distributed Rete algorithm Lana is an optimistic algorithm, allowing the different processing elements to run with minimal synchronization, informing a single central *Master Fact List* of changes to the working memory, and backtracking when the updates of the multiple replicated Rete engines conflict. Unlike in PARTE, the different entities running in parallel in Lana are fixed, allowing less flexibility to redistribute workload among the available processing units, and by splitting up the Rete graph, common subgraphs cannot be shared by multiple rules, requiring Lana to duplicate work where PARTE could reuse computations. Moreover, the optimistic approach generates a degree of nondeterminism with respect to run time which a real-time system like PARTE cannot risk to incur.

Yet other systems use hierarchical blackboard systems on which multiple agents concurrently work, and where every 'row' of nodes in the Rete network is reified as a different blackboard in the knowledge base's hierarchy. Examples of such systems are the Parallel Real-time Artificial Intelligence System (PRAIS) of Goldstein [9] and the Hierarchically Organized Parallel Expert System (HOPES) by Dai et al. [5]. Semantically similar, but not using the blackboard metaphor are for instance the parallel Rete system proposed by Gupta et al. [10]: they also acknowledged that having more than one token flowing through the graph at any one time could be opportune for the execution speed. In their paper, they proposed to give every node one or more internal threads of control. Their approach was conceptually very close to ours, but was not specialized for event-processing,

and supported conflict-resolution strategies like sequential Rete implementations. Because of this, they had to omit all conceptual optimizations which depend on temporal reasoning and many options for parallelism in the evaluation of the rules' consequents. Furthermore, their approach depended on hardware task schedulers to enqueue and dequeue node activations in a timely manner.

In general, previous approaches did not use the abstraction of actors like PARTE does. Their scheduling algorithms could not transparently handle nodes and the agenda, and they did not consider the nodes as self-contained elements. Data-locality was achieved in an ad-hoc manner, in Gupta's case expecting the presence of processor-local private memory in addition to the caches and main memory.

5.2 Real-Time Rete

Real-time execution characteristics can be achieved in two ways: *1*) by guaranteeing for every task that it completes in a known timeframe, and scheduling them such that they all complete before their deadline; or *2*) by assigning priorities to tasks, and allowing the system to preempt lower-priority tasks to ensure higher-priority tasks complete in time.

PARTE takes this first approach, and ensures that every action taken by the inference engine completes in time – unless unexpected load is put on the system by entities other than PARTE. The Parallel Real-time Artificial Intelligence System (PRAIS) of Goldstein [9] instead takes the second approach, dropping the matching of lower-priority rules to allow more important rules to be matched in time. Despite being considered a real-time system, PRAIS can only offer best-effort guarantees, as it is a distributed system depending on TCP/IP for communication.

Sequential implementations such as the self-stabilizing OPS5 production system by Cheng and Fujii [4] do offer actual hard real-time guarantees, but their approach is not viable for our problem domain: it requires lots of effort in the generation of the ruleset to provide fault-tolerance on top of the real-time guarantees. Their system is aimed at situations where failure is catastrophic and must hence be avoided at all costs. PARTE does not pose such severe restrictions on the ruleset, and only requires *tiering*.

6. Conclusions and Future Work

The presented PARTE inference engine implements a variation of the Rete algorithm using actor semantics for Rete nodes to achieve parallel execution. The system is designed for continuous gesture recognition which requires soft real-time execution guarantees and scalability on parallel systems. While PARTE utilizes the constraints of the domain to achieve these properties, it remains applicable to the broader domain of *Complex Event Processing*. This includes applications such as algorithmic stock trading and monitoring network security.

PARTE achieves the desired scalability and soft real-time guarantees by using a tiered architecture, lock-free queues, and an actor execution model to provide upper bound guarantees on the event matching in a Rete network.

PARTE is compatible with existing single threaded inference engines such as CLIPS from NASA. It has been used as a replacement for CLIPS in the core infrastructure of the multimodal Mudra framework [20]. PARTE provides the benefits of transparent parallelization of declarative rules as well as automatic event expiration.

Our preliminary performance evaluation used a number of microbenchmarks, kernel, and computational intensive benchmarks. The benchmark characteristics are representative for the gesture recognition and multimodal event processing context. In the current unoptimized state of PARTE, we achieve comparable performance to CLIPS. Furthermore, PARTE was demonstrated to scale on multicore systems with up to 8 cores, outperforming the inherently sequential implementation of CLIPS.

In future work, we will approach more efficient scheduling of the Rete nodes as well as exposing more parallelism opportunities by optimizing the Rete network. Support for an efficient implementation of a negation operator is planned as well.

Acknowledgments

Lode Hoste and Stefan Marr are supported by a doctoral scholarship of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen), Belgium.

References

[1] G. Agha. Concurrent object-oriented programming. *Commun. ACM*, 33(9):125–141, Sept. 1990. ISSN 0001-0782. doi: 10.1145/83880.84528.

[2] R. M. Akscyn, D. L. McCracken, and E. A. Yoder. KMS: a distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM*, 31(7):820–835, July 1988. ISSN 00010782. doi: 10.1145/48511.48513.

[3] M. M. Aref and M. A. Tayyib. Lanamatch algorithm: a parallel version of the retematch algorithm. *Parallel Computing*, 24(5-6):763–775, June 1998. ISSN 0167-8191. doi: 10.1016/S0167-8191(98)00003-9.

[4] A. Cheng and S. Fujii. Bounded-response-time self-stabilizing ops5 production systems. In *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000*, pages 399–404. IEEE Comput. Soc, 2000. ISBN 0-7695-0574-0. doi: 10.1109/IPDPS.2000.846012.

[5] H. Dai, T. Anderson, and F. Monds. On the implementation issues of a parallel expert system. *Information and Software Technology*, 34(11):739–755, November 1992. ISSN 09505849. doi: 10.1016/0950-5849(92)90169-P.

[6] A. A. El-Haj Mahmoud. *Hard-Real-Time Multithreading: A Combined Microarchitectural and Scheduling Approach*. PhD thesis, 2006. AAI3223132.

[7] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1): 17–37, 1982. ISSN 0004-3702. doi: 10.1016/0004-3702(82) 90020-0.

[8] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous java performance evaluation. In *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, OOPSLA '07, pages 57–76. ACM, 2007. ISBN 978-1-59593-786-5. doi: 10.1145/ 1297027.1297033.

[9] D. Goldstein. Extensions to the Parallel Real-Time Artificial Intelligence System(PRAIS) for fault-tolerant heterogeneous cycle-stealing reasoning. In *Proceedings of the 2nd Annual CLIPS ('C' Language Integrated Production System) Conference, NASA. Johnson Space Center*, pages 287–293, Houston, September 1991.

[10] A. Gupta, C. Forgy, A. Newell, and R. Wedig. Parallel algorithms and architectures for rule-based systems. In *Proceedings of the 13th annual international symposium on Computer architecture*, ISCA '86, pages 28–37. IEEE Computer Society Press, 1986. ISBN 0-8186-0719-X. doi: 10.1145/17407. 17360.

[11] A. P. Gupta, C. L. Forgy, D. Kalp, and A. Newell. Parallel OPS5 on the Encore Multimax. *Proceedings of the International Conference on Parallel Processing*, pages 271–280, 1988.

[12] T. Hammond and R. Davis. LADDER, A Sketching Language for User Interface Developers. *Computers and Graphics*, 29 (4), August 2005.

[13] T. L. Harris. *A pragmatic implementation of non-blocking linked-lists*. Springer, 2001. ISBN 3-540-42605-1. doi: 10.1007/3-540-45414-4_21.

[14] L. Hoste, B. Dumas, and B. Signer. Mudra: A Unified Multimodal Interaction Framework. In *Proceedings of ICMI 2011, 13th International Conference on Multimodal Interaction*, Alicante, Spain, Nov. 2011.

[15] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ, 2000. ISBN 0130996513 9780130996510.

[16] M. M. Michael. Safe memory reclamation for dynamic lock-free objects using atomic reads and writes. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing - PODC '02*, page 21, New York, New York, USA, 2002. ACM Press. ISBN 1581134851. doi: 10.1145/571826. 571829.

[17] R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*, page 267, New York, New York, USA, 1968. ACM Press. doi: 10.1145/1476589.1476628.

[18] D. P. Miranker. Treat: a better match algorithm for ai production systems. In *Proceedings of the sixth National conference on Artificial intelligence - Volume 1*, AAAI'87, pages 42–47. AAAI Press, 1987. ISBN 0-934613-42-7.

[19] K. Oflazer. *Partitioning in ParalleI Processing of Production Systems*. PhD thesis, Carnegie Mellon University, 1985.

[20] C. Scholliers, L. Hoste, B. Signer, and W. De Meuter. Midas: A declarative multi-touch interaction framework. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '11, pages 49–56, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0478-8. doi: 10.1145/1935701.1935712.

[21] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, volume 2, page 7, 2011.