CST report

Théo Rogliano

June 10, 2020

1 General information

Doctorat : Informatique et applications Ecole Doctorale : Ecole doctorale Sciences Pour l'Ingénieur Lille Nord-de-France Etablissement : Université de Lille Date de la 1ere inscription en thèse : 1 octobre 2019 (1 A en 2019) Directeur de thèse : Stéphane DUCASSE Sujet de thèse :

2 Thesis subject and objectives

Sujet initiale: Noyaux de langage multiples. La thèse va évaluer et concevoir une solution pour avoir plusieurs noyaux de langages executés côte à côte.

Maintenant: il s'agit d'explorer les differents mécanismes d'isolations et les differents systèmes de communication entre processus. Le tout afin de créer un système semblable à un micro système avec plusieurs petits noyaux qui peuvent être remplacer au besoin (par exemple si un de ces derniers tombent en panne).

3 State of the art

3.1 Erlang:

Erlang is a functionnal langage with ability to handle concurrency and distributed programming. The starter guide talks about function and modules. Modules are the closest entities to classes in the fact they encapsulate some attributes and some functions. In Erlang, each thread of execution is called a process. They are not called thread because they don't share data with each other (no race conditions). The Erlang built in function spawn is used to create a new process: spawn(Module, Exported Function, List of Arguments). It returns the procces ID have the spawned process. Message passing example between 2 process: (took from the guide)

$-module(tut15).-export([start/0, ping/2, pong/0]).ping(0, Pong_PID)- > Pong_PID!finished, io : format(''pingfinished\ n'', []);$

$ping(N, Pong_PID)- > Pong_PID!ping, self(), receivepong- > io : format(''Pingreceivedpong\ n'', [])$
$1, Pong_PID).$

$pong() - > receive finished - > io : format("Pong finished n", []); ping, Ping_P ID - > io : format("Pong received ping n", []), Ping_P ID!pong, pong() end.$

$start() - > Pong_P ID = spawn(tut15, pong, []), spawn(tut15, ping, [3, Pong_P ID]).$

Basically, they use an actor model for communication. The ! construct is used to send messages. The receive ... end construct is used to allow processes to wait for messages from other processes. Each process has its own input queue for messages it receives. New messages received are put at the end of the queue. When a process executes a receive, the first message in the queue is matched against the first pattern in the receive. If this matches, the message is removed from the queue and the actions corresponding to the pattern are executed. In the example, calling the process with pong 'Pong' and the process with ping 'Ping': 1: Ping sends a tuple ping, self() 2: ping, self() is put at the end of the input queue of Pong (the first element here) 3: Pong execute a receive and try to match pattern (here finished is a o match then ping, Ping PID is a match) 4: ping, self() is remove from the queue 5: Pong print and send a message to Pong 6: repeat

You can register to process in order to identify them and send them messages.

3.2 Pony:

Pony is an open-source, object-oriented, actor-model based langage. It is also statically and strong typed. Pony introduces the notion of behaviour which is an asynchronous function. When a behaviour is called he is not executed directly instead he is enqueued in the corresponding actor to be processed later . Pony diffentiates actors and classes by this fact, actors can have behaviour, classes can't.

Some properties about actors and behaviours:

An actor can process only one behaviour at a time, the processing of messages is sequential. An unknown number of behaviors can be process in the same instant but by different actors. An actor can only access his own state, never the one of another actor During his execution, a behaviour can't observe modification he did not do by himself.

To deal with data race, Pony forbids mutable shared data. This is ensured trough type annotation. There is 6 of them: iso, trn, ref, val, box and tag.

iso, trn ou ref references are mutable, they allow read/write/know the identity of the associated object. val or box references are immutable, they allow to read/know the identity of the associated object. tag reference is opaque, you can only ask for the identity of the object.

Two references are alias if they point to the same associated object. From a reference point of view, an alias is local is the other reference is in the same actor. Is is global if it is in a distant actor. For each annotation, they introduce rules about the type of aliases that can exist. Examples with a reference R:

If R is iso, the rule is: no aliases doing write or read can exist local or distant. Which means other aliases are annotated tag and you can only know the identity of the referenced object. If R is trn, global aliases doing write or read and local aliases doing read can't exist. Which means only tag and box aliases are allowed If R is tag, all aliases are allowed.

Put it in simple word, the type annotations ensure that you can't try to write at the same adresse through different aliases, for example.

3.3 Go

From https://golang.org/doc/

Go is expressive, concise, clean, and efficient. Its concurrency mechanisms make it easy to write programs that get the most out of multicore and networked machines, while its novel type system enables flexible and modular program construction. Go compiles quickly to machine code yet has the convenience of garbage collection and the power of run-time reflection. It's a fast, statically typed, compiled language that feels like a dynamically typed, interpreted language.

Go use 'goroutines' for concurrency. A 'goroutine' seems to be a thread. Here an example of code:

go func() time.Sleep(1 * time.Second) timeout ¡- true ()

the keyword go before the func means that this function will be done in a goroutine (not the main one).

In order to communicate between goroutines, you use channels . You can give: -the direction of the channel (only receive, only send, both) -the type of elements you will communicate -and the size of the buffer (number of elements) Examples:

chan¡- float64 // can only be used to send float64s ¡-chan int // can only be used to receive ints

To avoid data race, there seems to be an ownership system. Only the owner of the data can read and write on it. When you send the data to another goroutine trough a channel the ownership is passed. To cite them: 'Channels allow you to pass references to data structures between goroutines. you consider this as passing around ownership of the data (the ability to read and write it)' and: 'a Resource pointer on a channel passes ownership of the underlying data from the sender to the receiver.'

3.4 Scala

From: https://www.scala-lang.org/

"Scala combines object-oriented and functional programming in one concise, high-level language. Scala's static types help avoid bugs in complex applications, and its JVM and JavaScript runtimes let you build high-performance systems with easy access to huge ecosystems of libraries".

For concurrency Scala uses Future (https://www.scala-lang.org/api/current/scala/concurrent/Future.ht
and promise (https://www.scala-lang.org/api/current/scala/concurrent/Promise.html).

They define a future as: "A Future represents a value which may or may not currently be available, but will be available at some point, or an exception if that value could not be made available." With https://docs.scala-lang.org/overviews/scala-book/futures.html you can construct a future like follow:

$scala > valb = a.map(_*2)b : scala.concurrent.Future[Int] = Future(< notcompleted >$
)

The not completed is then Succes(84). To create a method that runs as a Future: Just pass a block of code into the Future constructor to create the method body.

A future returns an handle that can be used to register new callbacks (to post new things to do) when the future is completed, i.e., when the computation is finished. And since all this is executed asynchronously, without blocking, the main program thread can continue doing other work in the meantime.

I don't really know what to add. Do we already have article on futures that i can read ? (Santi ?)

3.5 Luna

The paper aims to solve the issues of data sharing, revocation, thread control and resource control on the language side instead of the system. (balance between control/sharing) It presents a solution for Java by extending the type system. The idea is "share data using special remote pointers, which have different types from local pointers" and having what they call a tasks model.

In the introduction is presented the deprecated API for Thread termination and the problem it had which are: -the wrong code get intreupted -malicious code can prevent rermination -malicious code can propagate -thread can get terminated while doing a read/write and corrupting an object -terminated code isn't reclaimed

Then we have the solutions to this kind of problems for: -DrScheme which adopt parent-to-child pattern. The parent is reponsible to cleanly terminate the childs. -J-Kernel where you can share only special capabilities which are revocable. You can perform remote method invocations. -The Java Community Process which only allows sharing copies. -KaffeOS (i will do the report soon too) which only allows to cummunicate with objects that do not contains overideable methods (primitive types and byteArray for example).

Finally we have the explanation about the remote pointers and the tasks model.

In their approach, "a single virtual machine will contain many tasks, each with its own objects, threads, and code. Inter-task communication is organized around the concept of a revocable remote pointer which is built into Luna's type system. Pointers from one task to an object in another task have a special type indicated by a tilde (e.g. "String-", "Hashtable-")". The revocabilty allows clean termination. When a task is terminated all pointers are revoked and deallocated. Remote pointers supports the same operation than local pointer but with a different semantic and an overhead. To handle the revocability you can use the Permit class. A remote pointer is a two words value composed of the pointer and a permit. "The @ operator converts a local pointer into a remote pointer:

Permit p = new Permit();

String s = "hello";

String- sr - s @ p;"

Operations on remote pointers perform a run- time access check if the pointer is not revoked.

It's still the application's duty to decide what to do in case of one of its communication partners becomes unreachable.

After that we have a view on how they implement their model.

They optimize the access to the remote pointers such as cache very similar to TLB (Translation lookaside buffer). We also have a quick view on they modify the GC to take into account the remote pointers. (check if the permit is still valid).

Follow by that we have benchmark on two application (extensible web serve, active cache) where they show they manage to reduce the overhead with the cache system varieting from 5 to 15

But array and field accesses were challenging to optimize, requiring extensive cooperation between the compiler and run-time system.

To finish there is a new comparison to related works already evoked and the article conclude in a bitter way, saying the problem seemed easy at first but was not in fact.

3.6 WriteBarrier

https://dl.acm.org/doi/10.1145/2991041.2991063 I wanted to do something similar to this article in order to have a 'lock' for different threads writing in an object. Thanks Pierre for the article.

The article talks about a write barrier per object in our VM with as little overhead as possible. The goal for the author is to have read-only objects. There is an extra check for every store into an object to see if the object is read-only (the barrier per se) in this case a callback is initiated to decide further action. Since there is extra instruction the author tires to minimize their impact.

The solution was implemented in three steps:

Enhancing the memory representation of objects to be able to encode their read-only state. (bit in the header to mark isReadOnly) Adding support in the execution engine to forbid read- only objects mutations.The objects are mutated in two main ways in the current virtual machine: -¿ By storing into one of their instance variable field (bytecode instruction). -¿ By performing a primitive operation that mutates object, such as at:put:. This is were our check will be added.

Adding support in the Pharo image to be able to use the new feature. (image side api, editing fallback code)

Note: some objects are hard or not meant to be read-only (Context instances, All objects related to process scheduling: the global variable Processor / the array of linked lists of processes (Processor instance variable) / ProcessLinkedList instances Process instances Semaphore instances, etc ...).

In the implementation part, we learn that he has done it for the JIT and to optimize the check there is trampolines. (can detail more if wanted but don't think it's usefull).

In the evaluation, he estimates that the code is 1.52He then did an edge case scenario where the overhead is 18

4 Progression and results

4.1 Isolation

In order to transfrom the vm with one interpreter to multiple threaded interpreters, the goal has been to 'group' the global variables used by the interpreter and that each interpreter uses its own set of variables.

The first part was already (almost) done in the vm. All the globals were already grouped in a structure called foo (we call it interpreterState) and all access to those globals were done by using a macro GIV(var) foo-¿var. So, instead of having a lot of globals we have one global, the interpreterState structure. Note: there is some other globals still, i added some into the interpreterState (like the primitiveFunctionPointer)

The second part, that each interpreter uses its own interpreterState has been done in 3 main steps:

1. For boucle
2. Local thread variable
3. Argument passing

The problem is we have a global that needs to be access by many threads.

For boucle: The interpreter being complex to read (low level, inlines) and touchy to change (one little mistake can rapidly break the whole thing) we started by modifying as little code as possible. The idea was to have an array of interpreterState of the size of the number of threads. Then each thread took one interpreterState by adding its ID in the interpreterState. To found back its interpreterState a thread had to go trough the array and found the interpreterState with its ID (a For boucle). Why is it simple to do ? We just need to add an array, add the for boucle in a function and change the macro to GIV(var) ForBoucleFunction()-¿var. Cons: it is slow as hell (10 times slower than the original interpreter).

Local thread variable: To speed things up we tought instead of having the interpreterState (or an array of interpreterState) as a global, the solution would be to have it in a variable local to the thread. The main difficulty here was that an interpreter uses already several threads (one for the interpreter, one for the heartbeat, one for the audio...) and all this threads should refer to the same 'local'. It takes to modify the code of this other components too. Cons: it is still slow (less than half the speed of the original interpreter).

Argument passing: The idea here is to add an extra argument directly at the creation of the thread which will contain the interpreterState. I omitted the fact that the code of the interpreter is generated. We modified the generation in order to add this extra argument first unused (already 30% slower than the original interpreter) and that's were i presented last benchmarks. Now to make this extra argument in used is more tricky. In the modified functions supporting the extra argument some of them are used in other files of the vm that are not generated (at least not by default). We can distinguish 2 cases, the plugins and the others (dunno how to call them). I started with the others, there are few and I modified them by hand. For the plugins, we started by thinking we could generate them but we did not find the generation for all of them and not everything were generated (for example in the FilePlugin, one file was generated but no the other). There is a lot of files to change (112) and a lot of different function calls to treat (1100). Since adding an extra argument is a refactoring, I tryied to handle this as a refactoring. In this order, I created an island grammar for the subset of C i needed to treat with PParser2. It allows to parse the plugins files(not perfect due to macro and some cases), create an AST for the subset of C and apply the refactoring with a visitor. In the end, after the refactoring there is still some compilation problem due to macro and the refactoring not being perfect. I corrected them and now the extra argument is on used everywhere and the impact is the same as when not in used (30 % slower than the original interpreter).

For the future, it could be nice to understand where the 30 % slow down comes from. I have a feeling that is something dumb and we can fix it easily. For example it could be the fact that i added the primitiveFunctionPointer in the interpreterState which is normal (each interpreter should ahve his own primitives) but complexify the primitiveFunctionPointer use.

4.2 Communication

Our goal through the channels is to enable communication between Process, OS Thread interpreters or another image. In our implementation, a channel is an atomicSharedQueue shared between Processes where one process can send objects or consume an object from the queue. In order to fully pass an object (and be thread safe) we added that the channel pass the ownership to one Process to the other in 2 different ways with: -A partial read barrier (Object¿¿become:). -A write barrier (Object¿¿beReadOnlyObject and Object¿¿beWritableObject)

Partial read barrier: we pass the ownership thanks to a become: when we send the object. the become remove all reference to the object from the sending process. Following the code of the send message:

send: anObject

— objectToSend — objectToSend := Object new. anObject become: objectToSend. queue nextPut: objectToSend

The process consumming the object in the channel gain a reference to the object and is now the only one with it. Pro: become: should be fast because become uses forwarder. Cons: some extra work are needed and should slow down the process if you want to give back the ownership

Write barrier: we start here by supposing all objects have an extra instance variable 'owner' which is the Process owning the object and are read only. Only the owner can write into IV of the object. When you send an object its owner become nil. When a process consume an object from the channel it become the owner. Pro: need less extra work than previous version. Cons: should be slow but (may be) could be optimize more. Some special objects like Array class or arrays do not support the become: message or to have an extra instance variable. So those objects cannot be send with a transfer of ownership with our implementation.

I think that will be a general problem for all classes with instances:

- if you change the ownership of a class, all its instances will point to the “revoked reference” =¿ so normal messages to the class should work =¿ but their instances do not! =¿ this is because this is a meta-object used by the VM

Future is to create example of application using channels and to compare both solution and enhance them if possible.

5 Plan for the next years

6 Formation followed

Catégorie : Formations numériques Gérer efficacement sa documentation avec Zotero - SPI (06 mars 2020 - 6 mars 2020) Crédit : 2 Total du nombre de crédits pour la catégorie Formations numériques : 2

Catégorie : Atelier technologique esug 2019 27th international Smalltalk Summer School (25 aout 2019 - 30 aout 2019) Groupe européen d’utilisateurs de Smalltalk, Cologne - Allemagne 40 heures Crédit : 20 enregistrées par : Ecole doctorale Sciences Pour l’Ingénieur Lille Nord-de-France. Total du nombre d’heures pour la catégorie : 40 h Total du nombre de crédits pour la catégorie : 20

Total participation : 40 heures / 2 modules

Total des Crédits de Thèse : 22

7 Professionnal project

Je ne me vois pas continué dans en tant que post doctorant ou enseignant chercheur. Je ne suis pas non plus fan de l’instabilité des start ups, du moins le fonctionnement américain et doit me renseigner sur le fonctionnement français. Je me vois plutôt devenir ingénieur de recheche ou dans une entreprise.

8 Difficulties

L’épidémie de Covid a rendu le travail un peu plus difficile.