Choose a language for a flynetting swarm

Tianyu Zhang, University of California, Los Angeles

Introduction

Flynetting swarm is a form of swarm robotics, which has been a hot topic for a long time. Swarm robotics is basically an approach that coordinates multiple robots as a system. The system contains many small physical robots and behave collectively. Moreover, the swarm robotics also require machine learning algorithm and cooperation algorithm. Currently the application is written by C++, but we are facing many different problems such as invalid address and race condition. In this article, we will discuss the possibility of rewrite the application in other programming languages. Specifically, we will be analyzing Python, Java, Go and C++ in terms of writing a swarm-robots application.

In order to determine the most fitful programming language for swarm robots, we need to know features used in swarm robots. In this article, we will need to compare the advantages and disadvantages of programming languages in parallelism, memory management and platform supports.

1. Parallelism

Programming a swarm of robots suggests a number of features found rarely in traditional software applications. Parallelism, which is the simultaneous execution of multiple subtasks, is a fundamental feature of swarms. Parallelism can be viewed merely as a performance boost. Moreover, swarms of robots provide a computational platform that requires software designed to operate over tens or hundreds of logical processors, rather than the two or four cores found in typical consumer products. In that case, we want to analyze the pros and cons of programming languages when they are written for parallel computing.

1.1 Python

If a Python thread performs a blocking operation, another thread will get the CPU and continue to run, while the first thread is blocked waiting. When the blocking event completes, the blocked thread will resume. So this is a good reason to implement multithreading in a Python script.

However, A big disadvantage of Python is Python's lack of multithreading support. Python uses a global interpreter lock, meaning that the synchronization of threads forces a one at a time execution. Due to the Global Interpreter Lock(GIL) that is present in Python, running code in parallel in multiple CPUs is not possible. The GIL ensures that only one thread is interpreting Python code at a time, so there isn't really a way to take full advantage of multiple CPUs.

Another way to address this problem is using asyncio. Asyncio multitasking is cooperative multitasking. The first advantage is that Cooperative multi-tasking is much lighter-weight than OS threads, so we can reasonably have millions of concurrent tasks, versus less threads.

1.2 Java

Different from python, Java does not have the restriction of GIL. Java is able to utilize multi-core processors in its threading applications to get more work done in the same amount of real time. Java utilizes parallel stream processing in Java 8 which makes It breaks problems into sub-problems which then run on separate threads for processing, these can go to different cores and then get combined when they're done. This all happens under the hood using fork/join framework.

However, one problem for parallel stream in Java is that it does not speed up the process. If we're already running multiple threads and we're using parallelStream() in some of them, adding more and more threads to the pool. This could easily turn into more than our cores could handle, and slow everything down due to increased context switching. Another problem is that java has a concurrent class for parallel processing. However, compared with Python which has asyncio library, it is much more tedious and difficult to implement parallelism in Java.

1.3 Go

In Go, parallelism is achieved by using Goroutines. Goroutines are functions or methods which can run concurrently with others methods and functions. They are very much similar like threads in Java but light weight and cost of creating them is very low. The advantage of Goroutine is that it has faster startup time than thread. And it comes with built-in primitives to communicate safely between themselves called as channels. Goroutines and channels are threads and queues in other languages. You can push a value into a channel and a goroutine can pull it off. Goroutines are also extremely cheap when compared to threads. They are only a few kb in stack size and the stack can grow and shrink according to needs of the application whereas in the case of threads the stack size has to be specified and is fixed.

Though go is designed mainly for concurrent applications, it still has some problems and drawbacks. Just because it gives a flexibility of creating a subroutine very simply, many times it becomes difficult to understand the working code structure. This is also in terms of functional programming vs object oriented. Go supports both and hence code many times can become harder to maintain unlike Java.

1.4 C++

Modern C++ has implemented many libraries and algorithms to make parallel programming easier. Moreover, C++ lets you write lock-free concurrent code so the compiler, with the help of some hardware features, can arrange for the compiled code to never attempt simultaneous changing of your shared data, while ensuring that different threads see changes when necessary. Moreover, C++ is more closed to OS, in that case, cut-edge thread research are often done with C++. For example, there are 22 C++ concurrency libraries. These libraries provide wide-range of support of concurrency.

However, there are still many problems. A major drawback is that concurrency features are not part of C++ and thus are not standardized. C++ needs library to achieve its parallelism unlike other languages which has parallelization features. Moreover, it is also tedious to achieve and maintain C++ code with parallelization.

2. Memory Management

Swarm robots has very little memory. In that case, a good garbage collection technique will possibly improve its performance. Garbage collection is the memory management process for objects in the heap. As objects are allocated to the heap, they run through a few collection phases – usually rather quickly as the majority of objects in the heap have short lifespans. A good GC techniques will reduce the overhead of concurrency. In that case, will improve the performance.

2.1 Python

Python handles automatic memory management by utilizing reference counting and garbage collection. Reference counting contains the number of times that an object is referred by other objects in the system. When the reference count of an object hits zero, the object is de-allocated. In addition, garbage collection occurs at scheduled intervals to remove objects with reference counts of zero from memory. The main advantage of the reference counting is that the program can be immediately and easily destroyed after they are no longer needed. A drawback is that implementation executes slower because it needs to update the reference count.

The reference counting is incredibly efficient and straightforward, but it can not detect reference cycle. Therefore, python comes up with a supplemental algorithm called generational cyclic GC, that specifically deals with reference cycles. GC will iterate over each container object and temporarily removes all references to all container objects it references. After full iteration, all objects which reference count lower than two are unreachable from Python's code and thus can be collected. A good feature of generational cyclic GC is that it is optional and can be invoked manually.

2.1 Java

Java implements automatic memory management as well, but doesn't use reference counting. It employs garbage collection. The garbage collection implementation lives in the JVM. Each JVM can implement garbage collection however it pleases; the only requirement is that it meets the JVM specification. The heap is divided into three parts. Newly created objects start in the Young Generation. The Young Generation is further subdivided into an Eden space, where all new objects start, and two Survivor spaces, where objects are moved from Eden after surviving one garbage collection cycle. When objects are garbage collected from the Young Generation, it is a minor garbage collection event. Then, objects that are long-lived are eventually moved from the Young Generation to the Old Generation. When objects are garbage collected from the Old Generation, it is a major garbage collection event. Metadata such as classes and methods are stored in the Permanent Generation. Classes that are no longer in use may be garbage collected from the Permanent Generation. In that case, during a full garbage collection event, unused objects in all generations are garbage collected.

The biggest benefit of Java garbage collection is that it automatically handles the deletion of unused objects or objects that are out of reach to free up vital memory resources. Moreover, Java’s garbage collection is faster than python because it could be done on another thread while the program is running.

2.2 Go

The operation of the Go garbage collection is based on tricolor mark-and-sweep algorithm. It can work concurrently with the program and uses a write barrier. This means that when a Go program runs, the Go scheduler is responsible for the scheduling of the application and the garbage collector. This is as if the Go scheduler has to deal with a regular application with multiple goroutines. The primary principle behind the tricolor mark-and-sweep algorithm is that it divides the objects of the heap into three different sets according to their color, which is assigned by the algorithm. The objects of the **black set** are guaranteed to have no pointers to

any object of the white set. However, an object of the **white set** can have a pointer to an object of the black set because this has no effect on the operation of the garbage collector. The objects of the **gray set** might have pointers to some objects of the white set. Finally, the objects of the **white set** are the candidates for garbage collection. when the garbage collection begins, all objects are white, and the garbage collector visits all the root objects and colors them gray. After that, the garbage collector picks a gray object, makes it black, and starts looking at whether that object has pointers to other objects of the white set. After that, the objects in the white set are unreachable and their memory space can be reused. Therefore, at this point, the elements of the white set are said to be garbage collected. Go's garbage collector is either optimized for lower latency or higher throughput. Tricolor mark-and-sweep algorithm is used in Go to lower that latency by running the garbage collector as a concurrent process. Moreover, Go's GC has been optimized to reduce the pause time which is lowered to <500 μs pause per garbage collection.

However, the garbage collection of go also has some drawback. One main drawback is that Go allows you to manually initiate a garbage collection by putting a runtime.GC() statement in your Go code. However runtime.GC() will block the caller and it might block the entire program, especially if the program is busily running many objects. When everything is rapidly changing, it will be very difficult for garbage collector to identify members in white, black and gray sets.

2.3 C++

C++ gives the programmer the provision of total control over memory management. This can be considered both as an asset and a liability as this increases the responsibility of the user to manage memory rather than it being managed by the Garbage collector. This concept is implemented with the help of DMA (Dynamic memory allocation) using pointers.

In this case, Compiler only removes the non dynamically allocated objects from stack segment once they are not in use. The compiler is not responsible for de-allocation of dynamically allocated objects. A user should always de-allocate them. It could be both advantage and disadvantage, if the user is familiar with dynamic allocation, C++ will be much more flexible compared with previous three languages. However, if the user is not very good at dynamic memory allocation, then the allocated objects could not be de-allocated which will lead to memory leak.

3. Platform Supports

Since we need to switch suppliers for our application, it is very important that our application can be used in multiple different platforms. We will discuss these four languages and analyze hardware that could support it.

3.1 Python

Python is a very common language used for machine learning and there is also many hardware that is able to support the language.

- System-on-Module from Coral
- Jetson Xavier NX from Nvidia
- Tensorflow Lite from TensorFlow
- TensorRT accelerator from Nvidia
- OpenVino from Intel
- MobileNetV3 and MobileNetEdgeTPU from Google
- Movidius VPU from Intel

Python can be largely used and applied on many different suppliers. It can let us switch suppliers when needed.

3.2 Java

Java is also a very popular language and can be supported by many different hardware.

- Tensorflow Lite from TensorFlow
- Movidius VPU from Intel
- OpenVino from Intel
- TensorRT accelerator from Nvidia

Though less platforms can support Java, we can still switch suppliers if write application in Java.

3.3 C++

C++ is a language that many developers use. It is more closed to operating system and also many hardware support it.

- System-on-Module from Coral
- OpenVino from Intel
- Movidius VPU from Intel
- Jetson Xavier NX from Nvidia
- TensorRT accelerator from Nvidia
- Tensorflow Lite from TensorFlow

From the list we can tell that similar to Python, many hardware supports C++ API. It also enables us to switch between many different products.

3.4 Go

Go is produced from Google and has been used mainly for parallelism. Since it is a relatively new language compared with three others. There are less supportive platforms for it.

- MobileNetV3 and MobileNetEdgeTPU from Google
- Movidius VPU from Intel
- OpenVino from Intel

Compared with others, utilizing Go as our language to develop application can be restricted sometime. But we can still switch between suppliers since we finally will only use one suppliers.

4. Conclusion

In this article, we compared four languages in terms of Parallelism, Memory management and platform supports. For parallelism, each language has its own feature, but I think the main goal of using parallel computing is to speed up the process and let applications perform cooperatively. In that case, Go is possibly a better choice for running parallel computing. For memory management, I think a good garbage collection technique will not only be beneficial to reduce overhead but also fitful for small memory in robot. Though I think Go's combination of auto GC and manual GC would be very efficient, it could be dangerous if user does not use manual GC properly and blocks the program. Therefore, python's GC is more reliable especially its reference counting plus manual generational cyclic GC. Finally, every language can be supported by more than two platforms. So the platform support will not be a restriction for language choice. However, I am still surprised by how C++ can be supported with many different hardware as python.

The comparison concluded from this article is far from settling down to the best language for swarm robots. To determine the language choice, there are still many other perspectives that need to be considered such as performance and networking. With a more thorough investigation of these four languages, we may be able to find the most fitful one for swarm robots.

References

https://medium.com/@tilaklodha/concurrency-and-parallelism-in-golang-5333e9a4ba64

https://jaxenter.com/java-8-problems-112279.html

https://medium.com/@bfortuner/python-multithreading-vs-multiprocessing-73072ce5600b

http://tleyden.github.io/blog/2014/02/06/why-use-go-when-you-can-just-use-c-plus-plus/

https://rushter.com/blog/python-garbage-collector/

https://hub.packtpub.com/implementing-memory-management-with-golang-garbage-collector/

https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html

https://coral.ai/products/

https://developer.nvidia.com/tensorrt

https://www.kdnuggets.com/2019/12/google-open-sources-mobilenetv3-improve-mobile-computer-vision-models.html