Concurrency in a distributed Gigapixel Image Viewer

Einar Kristoffersen
einar.kristoffersen@uit.no
University of Tromsø
Tromsø, Norway

Contents

1 Summary

As the current implementation of the Gigapixel image viewer at the TromsøDisplaywall is inefficient, hard to maintain and unstable, the need for a new design is growing. This technical report will describe the new concurrent and flexible design of the Gigapixel image viewer.

2 Introduction

2.1 Motivation

What is the motivation for this project?

2.2 Description of problem

What is the problem I am solving?

2.3 Contents of this report

What will this report contain?

2.3.1 Requirements...?

What is the requirements?

3 Technical Background

Talk about the technicalities... Golang, concurrency vs parallelism,

3.1 The Go programming language

Go is an open source project developed by a team at Google as well as other contributors from the open source community. It was designed to address the problems faced in software development at Google and became a great tool for engineering large software projects.

3.1.1 Why use Go?

As mentioned, one of the goals of this Project is to create a implementation that is maintainable. The current implementation of the image viewer is written by using a combination of several languages and as a result, it is hard to maintain the code. There was a discussion of which programming language would be best suited for the task. Both

Python and Go was solid candidates in order to create a maintainable implementation, but the decision landed on the use of Go to perform this task. Unlike Python, (C and C++), the Go programming language is build for concurrency. Go has the advantage of goroutines..

3.1.2 Concurrency build on CSP

Both concurrency and multi-threaded programming have a reputation for being difficult to use. Partly because of complex designs such as pthreads and partly too much focus on low-level details such as mutexes, condition variables and memory barriers. Higher level interfaces enables much simpler code and have been more successful in the past. One of the most successful models for providing high-level support for concurrency comes from CSP, by Tony Hoare. Channels... This is why the concurrency is build on the idea of CSP...

3.1.3 Goroutines instead of threads

Goroutines is a structure of behaviour that makes consistency easy to use in Go. The idea is to multiplex independently executing functions, coroutines, onto a set of threads. When a coroutine blocks, the run-time moves the other coroutines on the same os thread onto a different, runnable thread so they can keep running and not be blocked. The programmer sees none of this, which is exactly the point and makes it very easy to use the interface provided by this language. The result, also called a goroutine, can be very cheap. There is little overhead other than the memory for the stack, which is just a few kilobytes. In order to make the stacks as small as possible, Go uses resizable bounded stacks. Goroutines are given a few kilobytes, which in most cases is enough, but when it isn't the runtime grows and shrinks the memory for storing stack automatically. This allows many goroutines to run in a small amount of memory. Because of this, you can easy create houndreds of thousands of goroutines in the same address space. If a goroutine was just a thread, the system resources would run out at a much smaller number.

3.2 Concurrency

3.2.1 Concurrency vs. Parallelism

When people hear the word concurrency they often have a tendency to think about the concept of parallelism, a related but quite different term. In programming, concurrency is the organization of independently executing processes, while parallelism is the simultaneous execution of (possibly related) computations. When you run a program implemented by the use of concurrency, the executing processes is overlapping each other. This means that operations by different processes can work simultaneously, but they doesn't necessar-

ily start at the same time. Parallelism often refers to the technique of making programs fun faster by performing several computations in parallel, literally at the same time. This requires multiple computing units, while the concurrency often refers to techniques that makes a program more usable and requires at least one processing unit. Concurrency can be parallelism, but not the other way around. Concurrency more powerful than parallelism.

4 Design

4.1 Display Tool - Keyboard input

4.2 Draw/Blit

4.3 Data transfer and marshalling

4.3.1 Program flow

Flow charts

4.3.2 Data Flow

Whatever kind of input method is chosen, the received input will change the position variables kept at the master. Only when at least one of these variables are changed, they all will be marshalled/packed into a state. Every time a new state is paced it will be inserted into a buffer/queue (FIFO) where it will be sent out as a multicast to the slaves periodically.

We do this because we want the movement of the image to be as accurate as possible according to time with a minimal latency. We can afford to loose some states in this solution, compared to a solution where you send all states, obtain a huge latency over time and cannot afford loosing any states.

Variables: length of buffer/queue sendrate – how often should we let it send, FPS... empty the whole queue or just a bit?

Channel as buffer: Receivers always block until there is data to receive. If the channel is unbuffered, the sender blocks until the receiver has received the value. If the channel has a buffer, the sender blocks only until the value has been copied to the buffer; if the buffer is full, this means waiting until some receiver has retrieved a value.

when a channel is full, the sender waits for another goroutine to make some room by receiving you can see an unbuffered channel as an always full one : there must be another

groutine to take what the sender sends.

Input rate The input rate is at its peak when the the variables is constantly changed. This happens when the image is moving around on the screen and not by simply changing the zoomlevel. However, by how much the variables is increased or decreased during one loop count doesn't change the input rate in any way.

Variable change rate The amount of times the position variables is changed by the user input during one second will from now on be referred to as the "vcr" (variable change rate). Depending on the vcr is high or low, it might create a bottleneck in for of a queue before a state is multicasted to the slaves. As the master will send the current state of the position variables periodically, it will have a certain send rate. If the send rate is lower than the vcr, the send queue will be filled up and a latency will occur. As the queue grows longer, the time from receiving user input at the master to the corresponding state is sent from the master is growing larger. Not dealing with this problem will result in a growing latency and (show graph) from tests... To deal with this problem we introduce a empty policy saying that if the a state is tried to be inserted into a full buffer or the buffer grows too large, we empty the whole buffer.

Send rate

Blit rate

4.3.3 Network Flow

Sequential solution The sequential solution

Multicast solution

5 Implementation???

6 Tests/Evaluation

7 Results

8 Discussion

8.0.4 Why SDL?

SDL, OpenGL, Azul3D...

9 Conclusion

As we can see from the results and evaluation of the implementation and design ...