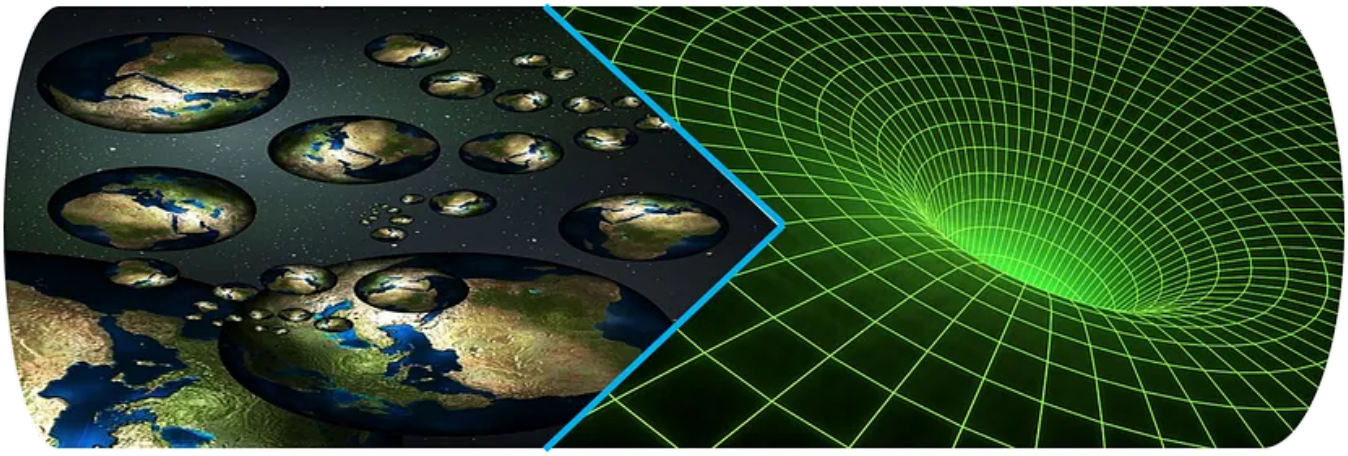
Fundamentals of High-Level Synthesis Part 3: From Concurrency to Parallelism (Map Pattern)

Mohammad Hosseinabady · Follow

5 min read · Nov 11, 2019



Images by Geralt and Johnson Martin from Pixabay

In the previous blog, I explained the difference between concurrency and parallelism concepts in high-level synthesis. Whereas the concurrency is a concept at the algorithmic or functional level, the parallelism is a realisation of the concurrency at the hardware level. Software programmers employ compiler tools to transform the algorithmic description into a representation suitable for hardware. Accordingly, in high-synthesis context, the role of HLS compiler is transforming the concurrency to parallelism to maximise the efficiency which can be throughput, latency, energy consumption among others. Therefore, the parallelism brings the performance and concurrency enables the parallelism exploitation. If designers cannot describe the concurrency adequately to enable efficient parallelism, then the desired performance cannot be achieved. Now, how does this transformation work? And how can designers describe the concurrency to facilitate this

transformation? To answer these questions, I try to explain different types of concurrency and parallelism patterns and suitable transformations among them.

How can we describe concurrency?

Algorithmic patterns are the fundamental techniques to describe and model the concurrency in an algorithm. Traditionally, patterns are defined as recurrent structures that can be used to describe, model, predict, solve or even implement problems, systematically.

Programming algorithmic patterns such as *map*, *reduce*, *scan*, *stencil*, just to name a few, can be used to describe the concurrency in an algorithm. In this post, I only consider the *map* pattern and try to explain how to implement that efficiently using HLS.

Map: This pattern receives an input dataset, applies a function (or a few functions) to its elements, independently, and generates an output dataset. There is no data or control dependency among these functions. Therefore, they can be easily modelled by a data dependency graph. Although the concurrency in this pattern seems trivial, finding the most efficient parallel implementation considering the hardware constraints and the resource cost functions can be tricky and challenging.

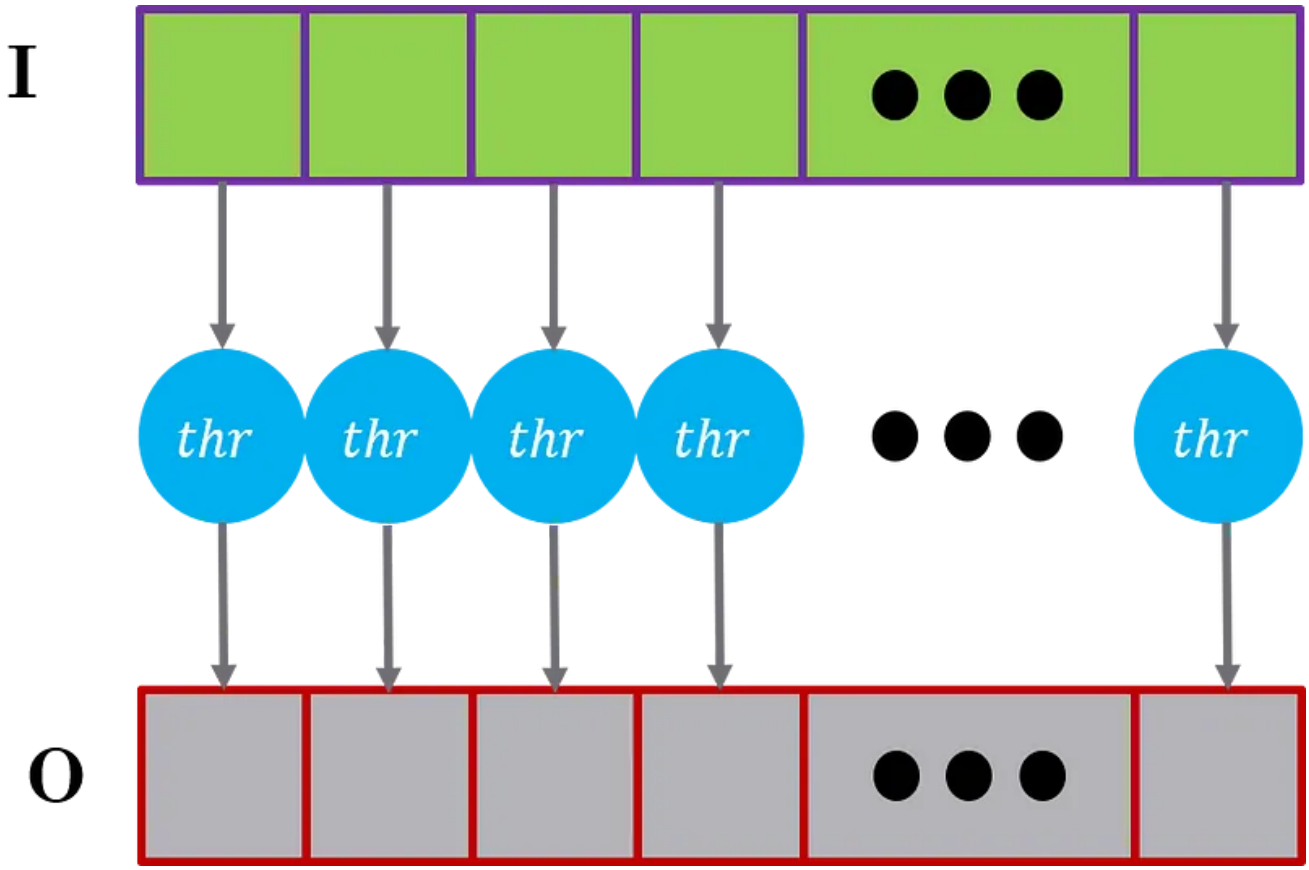
For example, let's consider the grey image thresholding as an example which is the most straightforward image segmentation algorithm. The naïve algorithm replaces each pixel of the image with black colour if its density is less than a fixed constant threshold *T*, otherwise with white colour. This algorithm can be modelled by a map pattern in which the following function is applied to each pixel independently, where *I(x,y)* is the input pixel density at the *(x,y)* location.

$$O(x,y) = thr(I(x,y)) = \begin{cases} 0 & \text{if} I(x,y) > T \\ 255 & \text{otherwise} \end{cases}$$

The corresponding C code as follows.

```
for (int x = 0; x < N; x++)
  for (int y = 0; y < M; y++)
    if (I[x][y] > T)
      O[X][y]= 0;
    else
      O[x][y]=255;
```

The following graph depicts the map pattern that applies the *thr* function to each element in the input dataset and generates an output element.



How can we describe parallelism?

Parallelism means running statements together. If two or more consecutive statements are independent, then the compiler schedules them together to be executed simultaneously if there are enough computation and memory resources available. For consecutive individual statements, the programmer does not need to

do anything except writing the code to increase the concurrency among them. In loop structures, the situation is a little bit complex that is explained in the sequel.

There are two main techniques two implements parallelism in iterative structures such as *for* loop: *loop unrolling* and *loop pipelining*.

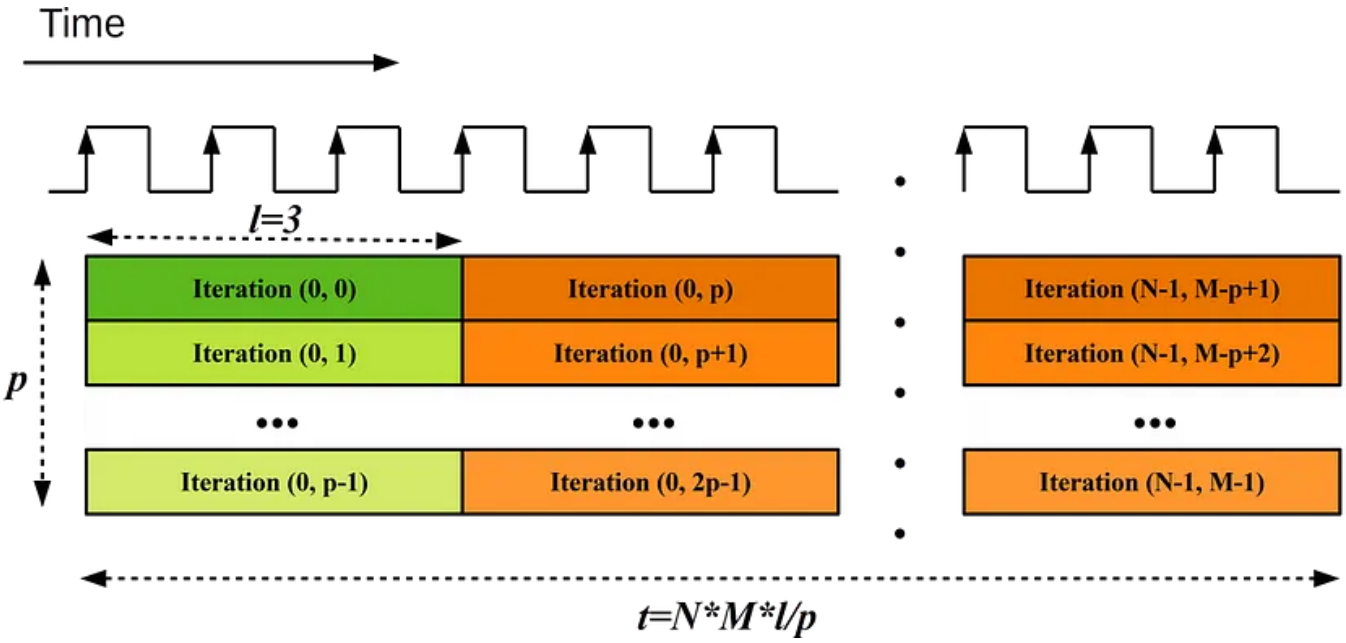
Loop unrolling

Loop unrolling is a common loop transformation technique in compilers to exploit the parallelism among loop iterations.

This technique enables the compiler to schedule loop iterations together. In the ideal case, a complete loop unrolling can provide the maximum parallelism if unlimited resources are available. However, because of limited memory ports and computational resources, partially unrolling loops is desirable. Finding the optimal number of loop unrolling is a challenge that varies for each problem.

Generally, the number of unrolled iterations (also known as loop unrolling factor) depends on the concurrency among loop iterations and the amount of resource available in the underlying hardware.

Let's consider the *map* concurrency pattern explained earlier. As all iterations are independent or there is the maximum concurrency, the number of FPGA resources defines the maximum loop unrolling factor that improves the performance. The number of memory ports available for reading and writing is usually one of the main factors that restrict the parallelism in the map pattern. If the memory has p ports for *reading* and p ports for *writing*, then unrolling the inner loop of the image thresholding code with a factor of p, improves the performance by a factor of p. Any unrolling factor more than p does not have any impact on the performance. Note that, there is a direct relationship between the number of memory ports and the maximum number of loop unrolling that can improve performance in the *map* patter.

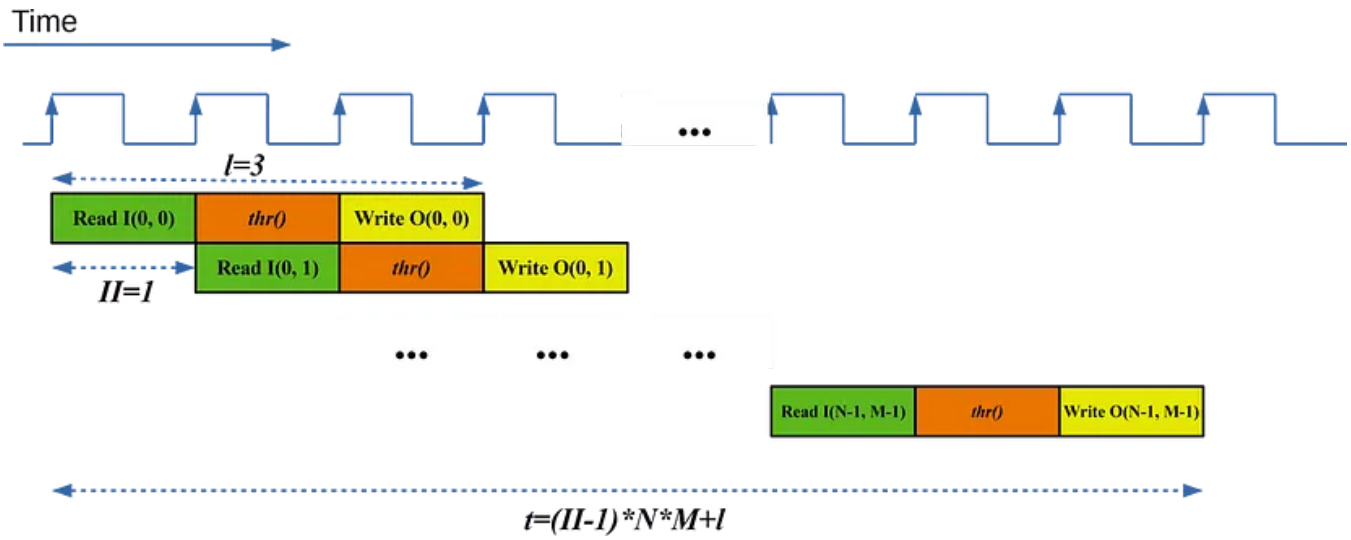


Loop pipelining

The loop pipelining is another technique to parallelise loop iterations. In this technique, a loop iteration is divided into multiple phases, and iterations are executed, having some overlap among these phases.

Source: homdor

Let's consider the image thresholding example if we divide each iteration into three phases: read *I*, run *thr()* function and write *O*. Then the pipeline timing diagram would be as follows:



The interested reader can think about using both loop-pipelining and partial loop-unrolling with the factor of p for the image thresholding example.

. . .

Originally published at http://highlevel-synthesis.com on November 10, 2019.

Programming | Software Engineering | Software Development | Hardware Accelerator | Hardware Startup

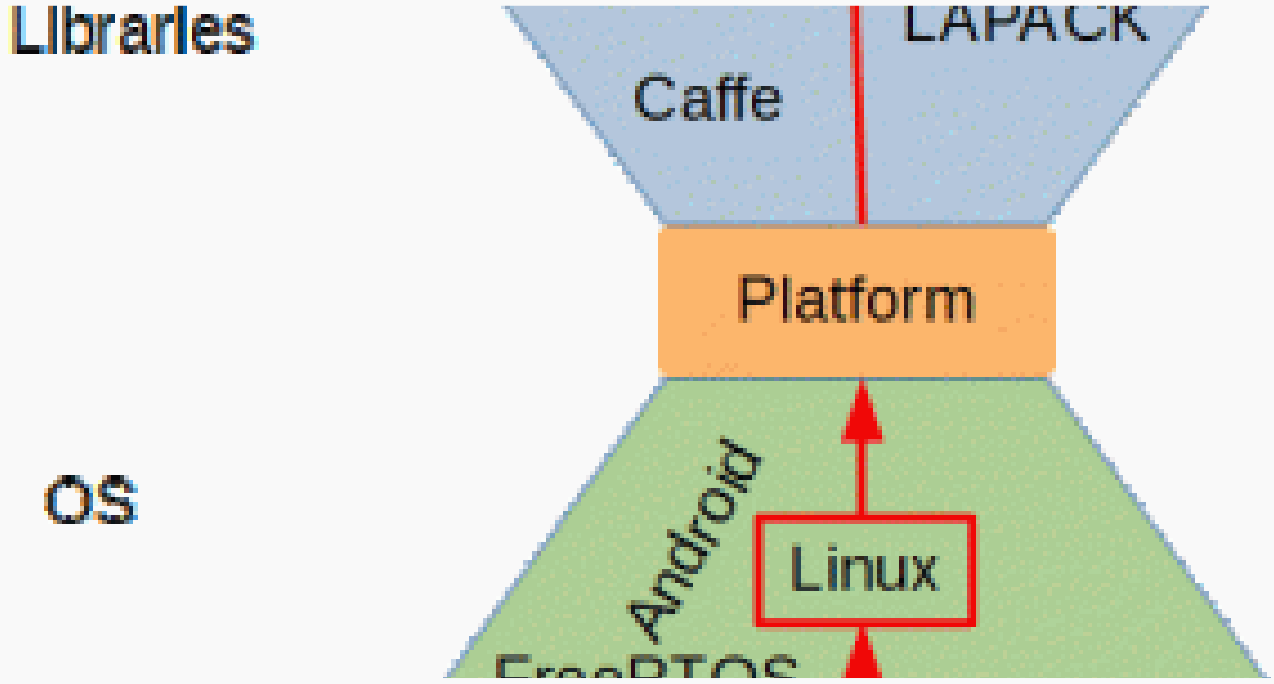
Follow

Written by Mohammad Hosseinabady

38 Followers

Designing digital systems and accelerating functions with HLS for FPGAs are fun.

More from Mohammad Hosseinabady

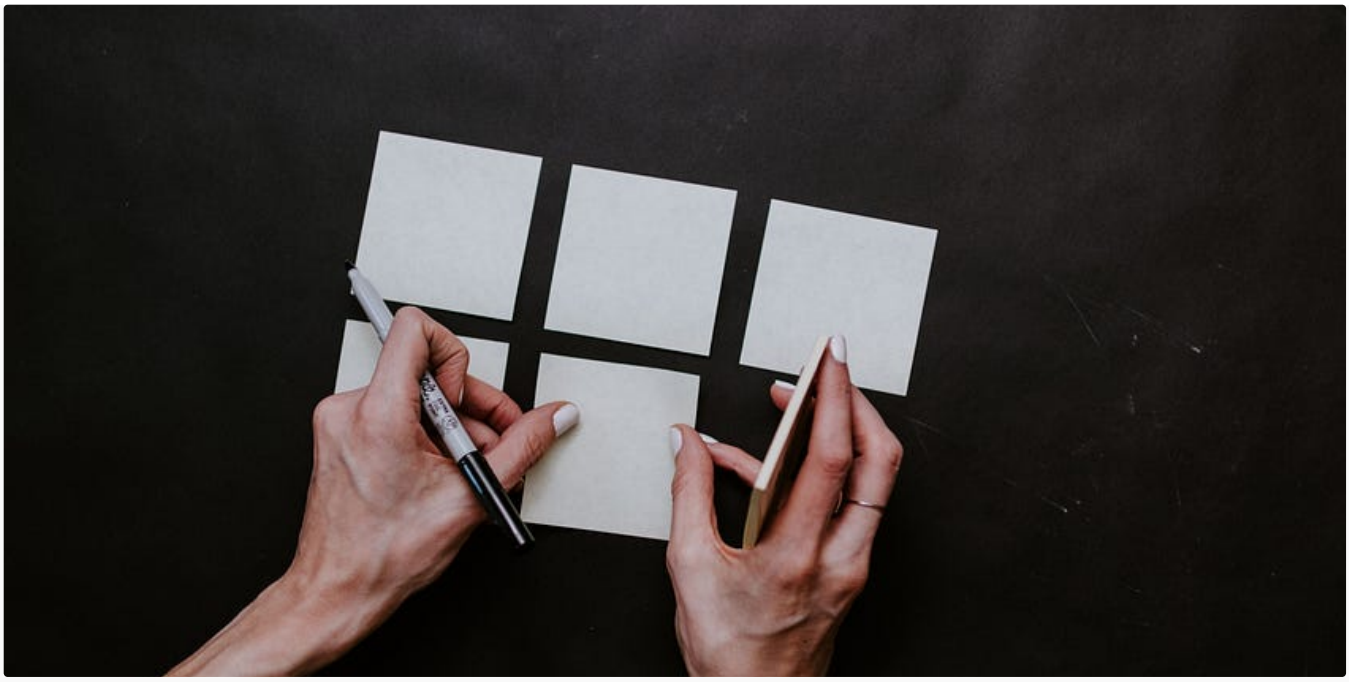


Mohammad Hosseinabady

Embedded Hardware Accelerator with Xilinx Vitis: Part 2: Create a Linux-based Platform

In the previous blog, I briefly explained the concept of platform-based design in the context of embedded FPGA. And the problems that it...

2 min read · Nov 15, 2019



Kurtis Pykes in The Startup

Don’t Just Set Goals. Build Systems

The Secret To Happiness And Achieving More

· 9 min read · Dec 21, 2022

21K 337

Martina D. in The Startup

11 Companies Making $1M+ That Are [Practically] Bedroom Businesses

Time to feed the ideation part of your brain with some real treats.

· 7 min read · Dec 31, 2023

1.8K 27

Mohammad Hosseinabady

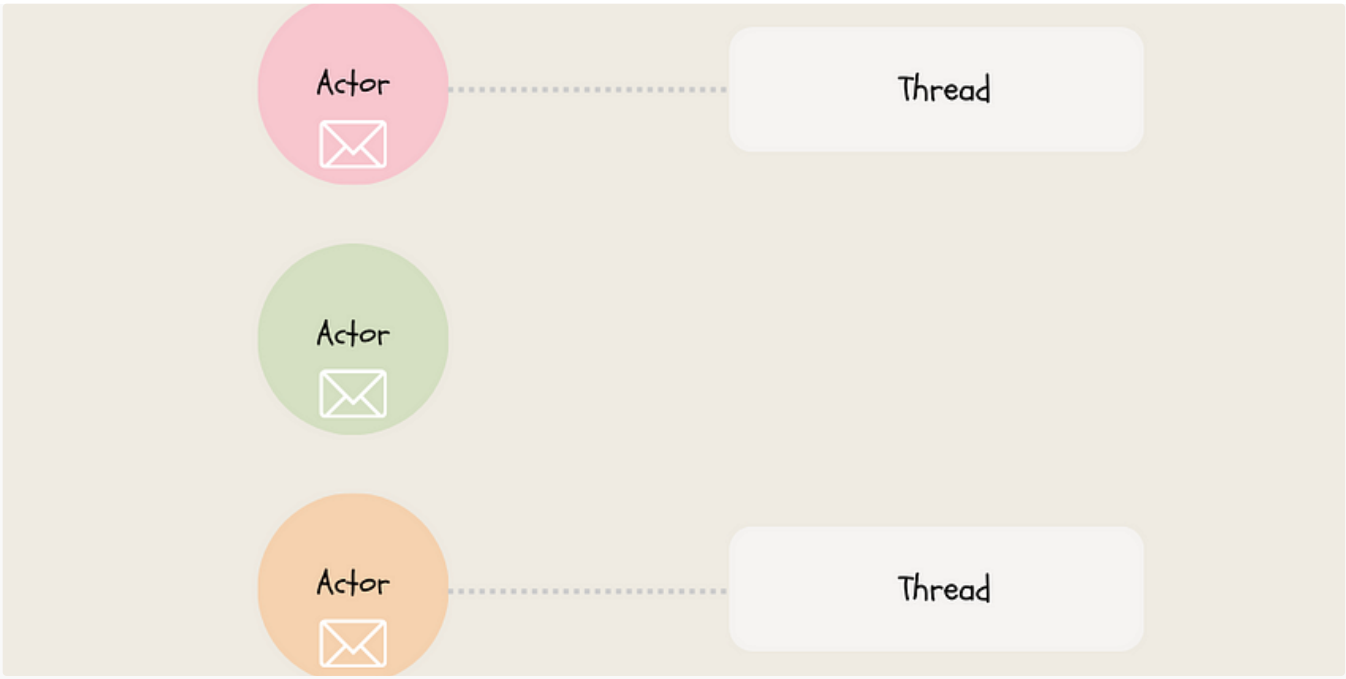
Embedded System HLS with Vitis: Communication with Software

· 6 min read · Dec 21, 2019

21

See all from Mohammad Hosseinabady

Recommended from Medium

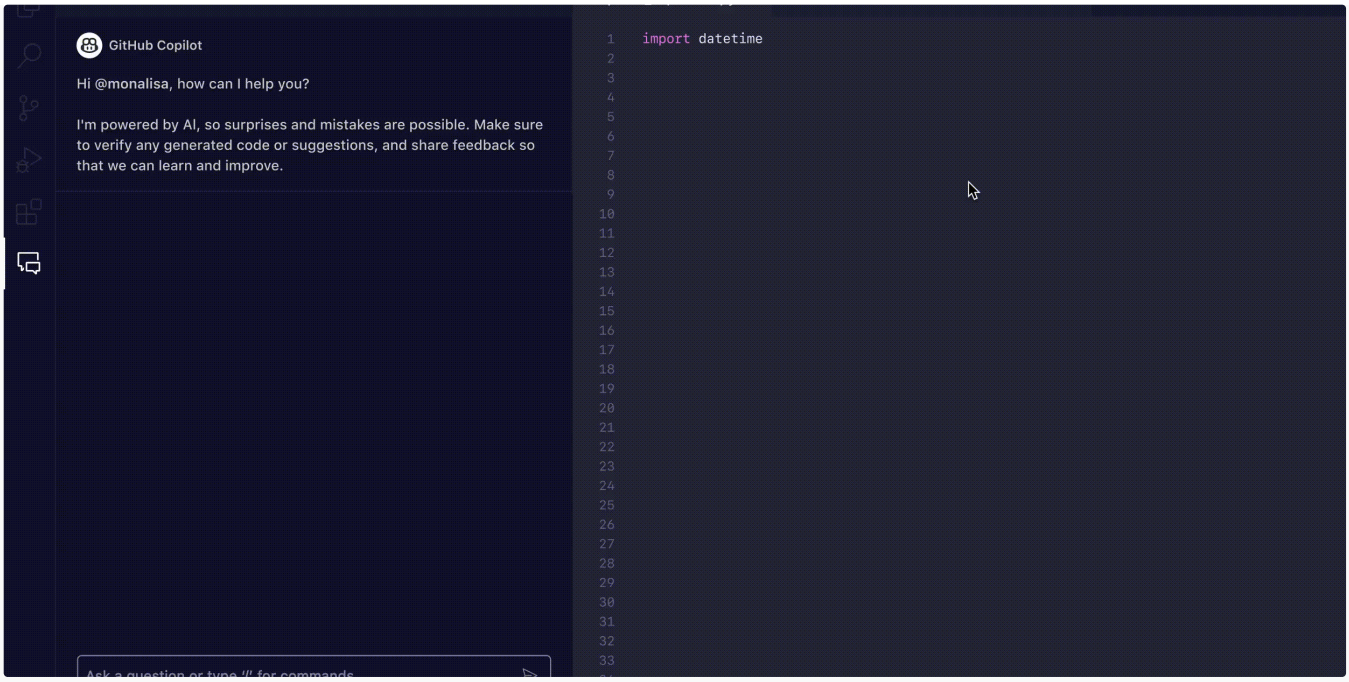


Nidhey Indurkar

How did PayPal handle a billion daily transactions with eight virtual machines?

I recently came across a reddit post that caught my attention: ‘How PayPal Scaled to Billions of Transactions Daily Using Just 8VMs’...

7 min read · Jan 1



Jacob Bennett in Level Up Coding

The 5 paid subscriptions I actually use in 2024 as a software engineer

Tools I use that are cheaper than Netflix

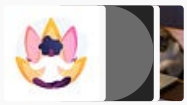
· 5 min read · Jan 4

2.6K 39

Lists

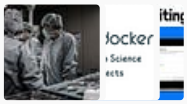
General Coding Knowledge

20 stories · 801 saves



Stories to Help You Grow as a Software Developer

19 stories · 718 saves



Coding & Development

11 stories · 385 saves

Leadership

41 stories · 201 saves

Companies have found that "Agile", as it is sold, delivered, and explained to them, does not work. You can blame them if you like, or you can blame the Agile community for not packaging the right kinds of learning and support. But regardless, "Agile" as we know it is dead. And Scrum will go with it.

But companies still need _agility_. Real agility. That has been our focus.

Real agility is mostly behavioral, and in particular, it is driven by the behaviors of leaders. Leadership is the big glaring hole in the Agile Manifesto. It is like trying to make concrete without water. No wonder "Agile" did not work.

That's why Agile 2, which reimagined what "Agile" should have been,

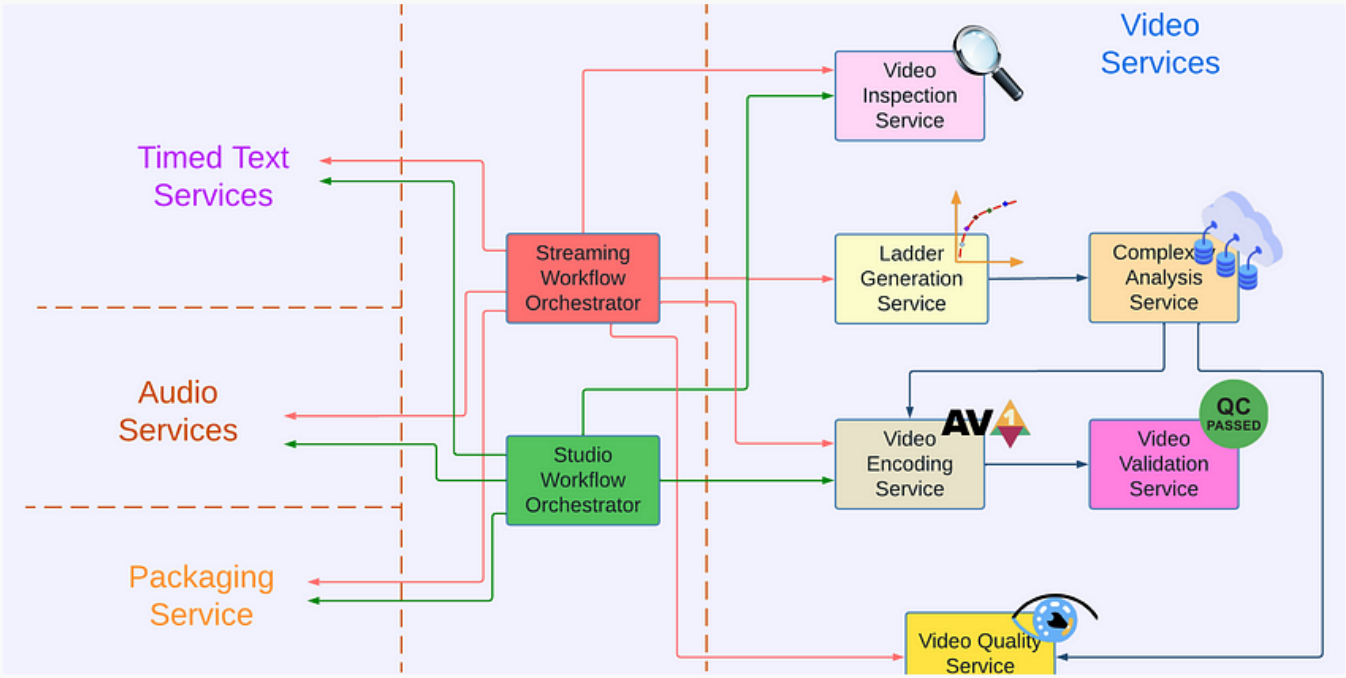
Tamás Polgár in Developer rants

Agile has failed. Officially.

Either I'm a gifted oracle, and all of my friends are, or Agile really was just a stupid idea to begin with. After many years of agony...

2 min read · Dec 3, 2023

13.1K 407



Netflix Technology Blog in Netflix TechBlog

Rebuilding Netflix Video Processing Pipeline with Microservices

This is the first blog in a multi-part series on how Netflix rebuilt its video processing pipeline with microservices, so we can maintain...

9 min read · Jan 11

1K 23

Benoit Ruiz in Better Programming

Advice From a Software Engineer With 8 Years of Experience

Practical tips for those who want to advance in their careers

22 min read · Mar 21, 2023

13.5K 251

Vaishnav Manoj in DataX Journal

JSON is incredibly slow: Here's What's Faster!

Unlocking the Need for Speed: Optimizing JSON Performance for Lightning-Fast Apps and Finding Alternatives to it!

16 min read · Sep 28, 2023

12K 141