CS 1632 – Deliverable 4: Performance Testing

William Stephenson, Grant Kelly

willgs, WGrantKelly

https://github.com/willgs/D3.git

**Issues Faced**

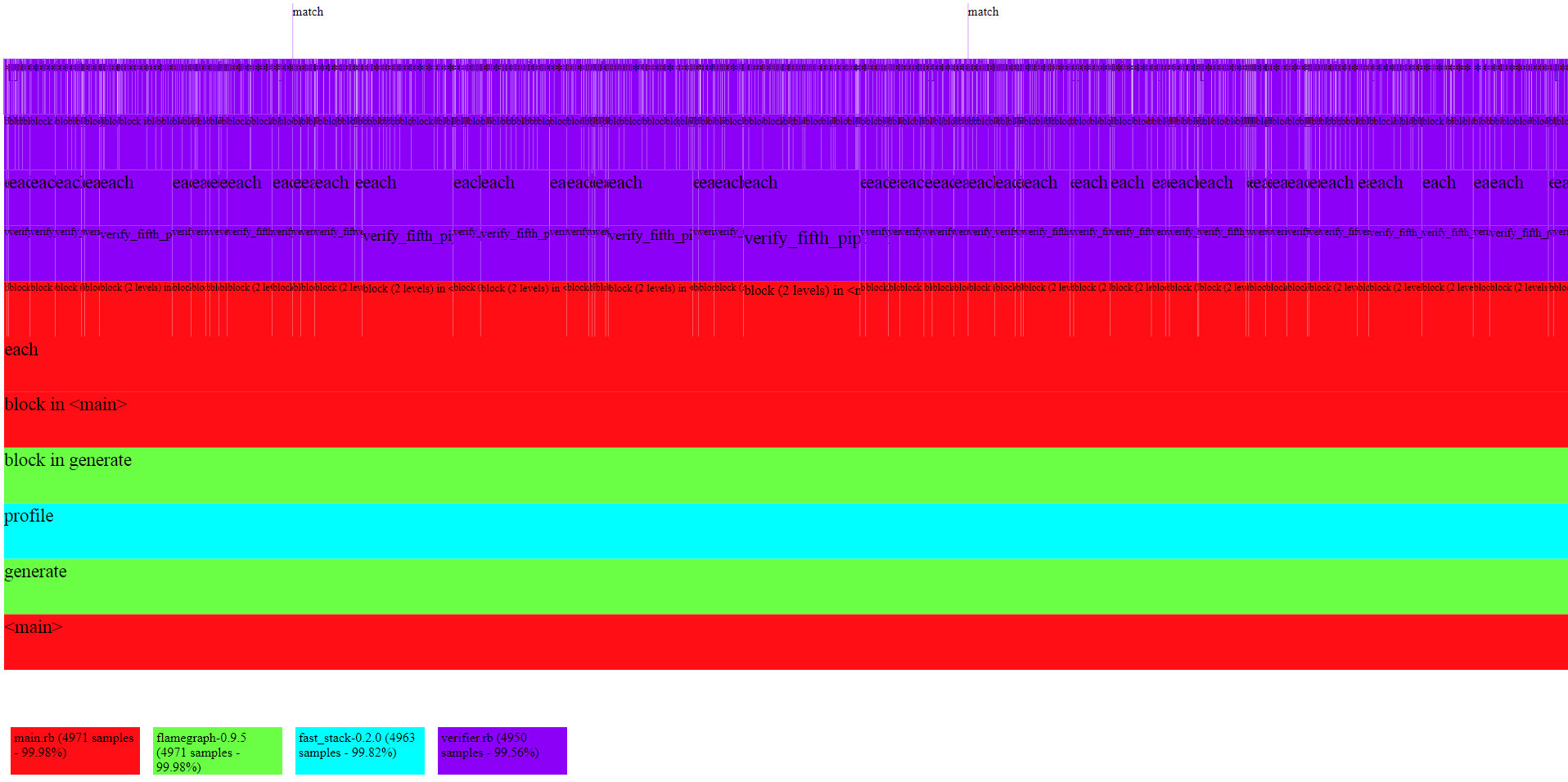
While writing this code I one of the first problems me and my partner had was understanding the how to implement the functionality of the deliverable. Even though this project was a very simplified version of blockchain verification, it was still a new and somewhat complicated process for us to implement. It took several re-reads of the billcoin.md file and about a half an hour of discussion between me and my partner to fully understand what we were implementing and then about another hour to determine the structure of our program and begin dividing the work.

The way in which structured our code impacted the failure cases we considered. Because we were using white box testing in this way, we knew that all of the arguments passed to the methods that verify the data in between pipes would all be strings. We did not need to check for bad data types, only bad data. We tested for invalid syntax in each pipe set, as well as invalid data. For example, in the time pipe set, we tested for invalid characters (anything non-numeric) as well as out of order times.

Based on the flame graph, we could see that raising x^x was taking a significant amount of computational effort. Mathematically, this is to be expected and there is little that can be done to optimize this out.

Due to time constraints, we were not able to optimize our code to any degree. However, given the data from the flame graph, we can see that we would have focused on the verification of the fifth pipe set, which makes sense because that is the function in which hashing actually occurs. There are some interesting mathematical shortcuts I would have tried if I was able to contribute more time to this. One potential idea was to see if our character value equaled 7 or 3, since that would have eliminated the need to do one of the power operations twice. However, it is also just as likely that this comparison of our value to 7 or 3 would have taken more time than that one skipped mathematical operation would save.

**Flame Graph Before Optimization**



**Stats**

Hotspots: verify\_fifth\_pipeset.

(In particular, the block.each loop and the \*\* operations take the most time.)

Mean run time: 49.843 Seconds