Parallel Proving of zkVM Programs On Computationally Weak Nodes; A Risc0 and Swarm Case Study

Reza Hasanzadeh
Wholesum Network
rezahsnz@proton.me

July 18th, 2024

> Do't trust, Verify.
>
> *Crypto Community*

1 Introduction

zkVMs provide execution correctness guarantee for programs they run. The output of a zkVM program is accompanied with proofs that allow independent entities to verify if the execution was as expected. The guarantee comes with expensive costs though. A typical zkVM program is at least 10x slower compared to its unverified native execution. There are important situations however that justify the incurred additional costs. An ETH L2 zkRollup is a scalability example where the correctness guarantee for transactions is secures the movement of billions of dollars. Verifiable computing, bearing significant computational overhead, requires strong (local) computers. And by strong, we mean high-end GPUs or server-scale CPUs at a minimum. Given the in-feasibility of owning such devices for everyone and the hope to capitalize on this booming technology, there has been an influx of (decentralized) prover/GPU markets where users are incentivized to rent their computers out in exchange for money. Decentralized shared sequencers, general verifiable computing, and zkML are among the most sought subjects when looking for justifications to run zkVM compute networks. There also exists general compute networks where strong nodes are provided, but no verification tools are available and users have to customize the computation pipeline themselves in order to benefit from verifiable computing. These networks are usually designed for scientific computing or rendering workflows and are among the oldest compute networks in existence.

Decentralized storage networks(dStorage from now on), e.g. Swarm, are another type of distributed computing networks where the sole goal is to rent storage space out and make money. Compared to GPU markets, these networks rely on humbly weak nodes to serve users. The typical computational burden for a dStorage node is to wake up from time to time and sync itself with the network.

This leads to relaxed minimum system requirements for nodes to join dStorage networks. Nodes often sport fast SSDs, but old CPUs, and unsurprisingly lack GPUs. Even with such weak computational power, they are idle half of the time. This situation raises an interesting engineering question: is it possible to push the overall node utilization CPU-wise to 99%? and if yes, what kind of computing we expect to carry out on a set of computationally weak machines? It turns out that there might be an opportunity to form a network off such storage-maxi nodes and task them with proving zkVM programs in a parallel manner.

2 Risc0

Risc0 is a zkVM that enables arbitrary(well, to some extent) rust programs' execution to be verifiable. A given Rust program first gets compiled into a RISC-V ELF binary whose entire execution trace is then fed to the zkSTARK circuitry of Risc0 to produce a prove-ready executable program. On the code level, Risc0 divides a zkVM application into two parts: *host*, the native untrusted, and *guest*, the provable enclave. The host loads and prepares the environment(I/O handling, ...) for guest to be proven. Execution-wise, the host and the guest are unified into a single Rust executable. Subject to requirements, developers usually customize host's code to relay out generated proofs for independent verification. The actual proofs and the public output of the guest is wrapped inside a small object called *receipt* that is the result of proving the guest. Receipts contain everything verifiers need to approve any guest's execution correctness.

Babybear field[1] is used for proof generation in Risc0[Bruestle et al., 2022]. This choice imposes caps on how many instructions can be proved in one go. The limit is set to be 2^{24} *cycles*[2]. To overcome this limitation, Risc0 came up with the idea of *continuations* to make infinitely-sized guests possible. Continuations build on the idea of dividing guest into *segments* with the segment size capped at 2^{24} cycles. With continuations, therefore, there is no limit on the number of segments and thus guest program size can grow to infinity. Large programs are usually composed of several segments and to prove them, each segment is proved individually and the final proof is obtained once all segments are proved. Developers can customize segment size to lower limits. The acceptable range is $2^{13} \approx 8k$ to $2^{24} \approx 16m$ cycles. Picking a large limit results in fewer segments but big memory requirements, while lower limits introduce excessive memory mapping to keep things consistent. So, choosing the right limit needs proper justification. Table 1 lists segment size limits and their memory requirements.

3 Swarm network

Swarm network is a dStorage platform. Its main goal is to provide a p2p storage solution with privacy, resiliency, and availability promises[The Swarm Team, 2021]. Being a network, Swarm, is formed by nodes that provide their storage space in exchange for money. Content that needs to be stored, is divided into 4 KB chunks, encrypted, and sent to nodes for storage. A JPG file, for example, ends

[1] $p = 2^{31} - 2^{27} + 1$

[2] A cycle is equivalent to one or two RISC-V operations.

Segment size *cycles*	**Minimum required memory** $\approx$ *amount*
$2^{13} \approx 8k$	64 MB
$2^{14} \approx 16k$	128 MB
$2^{15} \approx 32k$	256 MB
$2^{16} \approx 64k$	512 MB
$2^{17} \approx 128k$	1 GB
$2^{18} \approx 256k$	2 GB
$2^{19} \approx 512k$	4 GB
$2^{20} \approx 1M$	8 GB
$2^{21} \approx 2M$	16 GB
$2^{22} \approx 4M$	32 GB
$2^{23} \approx 8M$	64 GB
$2^{24} \approx 16M$	128 GB

Table 1: Memory requirements per segment size

up being stored by hundreds of nodes(each node stores some chunks) in a redundant manner to maximize protection against single points of failure. Swarm follows content addressing model of storage where CIDs[3] that are derived from a *merklized* variant of chunked layout of content are used for retrieval; *lose CID, adios content.*

Swarm, puts relaxed requirements for node participation. This is typical of all dStorage platforms as well. To join Swarm, a node needs to have at least 4 GB of memory, a dual core CPU and is encouraged to sport a SSD drive. This setup is more than enough to join dStorage, because the act of storage is not computationally intensive. A node's typical contribution is to wake up from time to time, do some syncing(store new content, prove the storage of old content, house keeping, ...), and go back to sleep. It can be argued that nodes are expected to be idle half of their dStorage lifetime[4].

As of July 2024, Swarm has more than 15,000 active nodes. Ideally, 15,000 half time idle nodes translate to 7,500 completely idle nodes. Given the minimum system requirements to join, the average TFLOPS of a Swarm node is estimated to be around 0.2 TFLOPS[5]. The combined total computational capacity of Swarm, therefore, hovers around 3,000 TFLOPS. So, when Swarm is viewed as a mere computational network, it can emulate 35 RTX 4090[6] GPUs, while when treated as a hybrid storage-compute network its computational capacity is halved and can only emulate 17 4090s. This is an idealistic deduction though, and the amount of latency and noise typical of p2p networks drags the actual value much lower.

4 Parallel proving

Risc0 zkVM programs require upper middle or high-end GPUs for production-ready fast proving. Mobilization of such hardware into a network form is complex. There needs to be a strong set of monetary incentives for participants

[3]Content Identifier

[4]This is only an observation and we have no data to back the claim.

[5]The value is obtained by halving the compute power of the GPU of a decade old example CPU, e.g. Intel HD 5500

[6]NVIDIA RTX 4090, with 83 TFLOPS of compute power, is indeed a heavy duty GPU.

to put their gadgets at work. No demand and the network faces exodus. Low demand and starvation is a problem. It is only when the demand is high that the network reaches a sustainable state. dStorage, on the other hand, is easier to approach because nodes are already *storage-maxi*: have some tasks to do, and have accepted staying half idle as their fate. But given the demanding nature of zkVM proving, assigning a functional role to (computationally weak) storage-maxi nodes seem euphoric unless a robust distributed and parallel proving design is employed.

The hack is to limit the segment size of the guest to a value that is stress-friendly to Swarm nodes. Here we assume that nodes have at least 8 GB of memory. Of all segment size limits shown on Table 1, values up to 2^{19} seem to be good choices because the amount of memory required for proving respective segments stay below 4 GB. Nodes are already running an operating system with resident services so they would need some free memory to stay responsive to new events. The 4 GB limit guarantees that nodes stay storage solvent while being open to the inbound zkVM prove tasks. We have experimented with various guest program sizes and how they react to different segment size limits in terms of overhead and other side effects[Hasanzadeh, 2024]. Table 2 is borrowed from the report where a 90M cycles program gets segmented with various limits and then proved.

Segment size	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
# Segments	16,808	2,174	801	355	168	82	41	20	10	5
Total cycles	551m	142.5m	105m	92.9m	87.8m	85.5m	84.4m	83.9m	83.9m	$\approx 83.9m$[a]
Induced inflation	6.566x	1.698x	1.252x	1.107x	1.047x	1.019x	1.006x	1.000x	1.000x	1.000x
Prove time	14s	28s	58s	119s	249s	534s[b]	-	-	-	-
Sample size	0.1%	2.5%	5%	5%	5%	3%	-	-	-	-
Blob size	19kb	38kb	67kb	133kb	265kb	529kb	1.1mb	2mb	4.1mb	6.4mb

[a] Exact value: 83,886,080

[b] low value is due to small sample size

Table 2: Segmentation of a 90M cycles guest

Focusing on the 2^{19} column, as a sweet spot, we need to prove 168 segments where it takes 249 seconds to prove each one on a 10 year old laptop[7]. Now if we were to set the segment limit to the highest value(i.e. 2^{24}) and prove it on the same machine, we would have needed 128 GB of memory and had to wait for 15 hours. Segmentation of the guest into many segments and *paralleled proving*, if feasible, results in up to 200x speedup compared to solo proving. In reality, however, those who need to prove such big zkVM programs almost always rent strong cloud computers.

To prove the 90M cycles example program of Table 2 in 249 seconds via Swarm network nodes, we need to have at least $168 \times 5 = 840$ ready nodes. This redundancy is due to the fact that p2p networks are a hotbed of noise. And by setting signal to noise ratio to 1:4(four time wasters nodes for every get-the-job-done one), we can effectively guard against time waster. Swarm with its fleet of 15,000 nodes is thus capable of proving 17 90M cycles programs simultaneously. 17 is obtained when latency is assumed to be 0 seconds, while on reality it is way longer. The full capability of Swarm for zkVM proving

[7] Lenovo Thinkpad e450: Intel Core i7 5500U(2C 4T), 16GB RAM

has yet to be measured and the design of the parallel proving protocol is an important factor here.

Another point of interest on Table 2 is the *Blob size* row. When segmenting a guest in Risc0, each segment is serialized into an independent binary object. Once a segment is ready, the protocol sends it to the network and waits. Each prove result when available, needs to be verified. The final proof is the aggregation of all proved segments and this is what the protocol sends to the client as the overall result of the whole prove job. A typical flow of a prove job is as follows:

I Segment the guest into G items

II Store all segment blobs on Swarm and obtain CIDs

III Dispatch segments(CIDs actually) to the network for proving

IV Wait for all segments to be proved and verify them individually

V Combine all proofs into a final proof and verify it

VI Deliver the result to the client

Note that the redundancy guard introduces no additional storage overheads. It is only the actual proving that may be carried out excessively. The segment blobs are stored once and retrieved many times. It is the CID of the blobs that are sent around and this is by design. Sending large objects is not feasible in p2p settings. The 2^{19} segment size limit on Table 2, for example, results in 168 blobs of each 265 KB in size. A message with 265 KB as payload size is prohibitively large by any standard, so we restrain messages to only carry CIDs.

Choosing smaller segment limits is a possibility too. But we should keep in mind that as the number of segments grow, the quality of the parallel proving algorithm may reduce. Maintaining such big proofs as 355 segments in the case of 2^{18} segment limit size on Table 2 is a complex stress test for any network. The actual values for global network parameters, therefore, need to be tested in battlefield and evolved into a sustainable set of values.

5 Applications

In this section we briefly review some of the potential application areas where parallel proving may fit in. These areas are among the top trends within the verifiable computing community.

5.1 ETH L2 sequencer decentralization

L2 zkRollups aggregate transactions, prove their integrity, and post generated proofs to the ETH network for verification and inclusion. This method is a basic practice for scaling the ETH network. zkRollups prove locally, meaning that they use centralized compute services to generate proofs. The entity responsible for this job is called a *sequencer*. The sequencer's role is to aggregate transactions and generate integrity proofs. Linea's sequencer, as a L2 zkRollup example, relies on a *hpc6a.48xlarge* AWS instance to prove transactions. This cloud computer has a 96-core AMD EPYC CPU(up to 3.6 GHz) and 384 GB of

memory. It takes 5 minutes for the machine to prove a set of transactions that occupy a 30M gas mainnet block. The prover is not open-source as of today and thus we do not know much about its inner workings.

Linea's prover is not decentralized either. We can provide an alternative decentralized prove mechanism if our parallel proving mechanism gets the job done in less than 5 minutes. The worst case scenario for us is to assume that the centralized prover needs all of the 384 GB of memory space. Extrapolating Table 1's values, 384 GB of memory is mapped to a 50M cycles guest program. Add STARK to SNARK transformations, and we might hit 90M cycles. For the sake of extremism, let's assume that we are going to prove a 90M cycles guest with 1:4 compute redundancy ratio assumption. We detailed a similar scenario in the previous section. There we deducted that to prove a 90M cycles program in less than 5 minutes(we need to be lucky too), we would need 840 nodes if we limit the segment size to 2^{19} cycles. To prove even more quickly, we have no choice but to reduce the segment limit to 2^{18} cycles. With this new limit, though it takes 2 minutes to prove, we are going to need $355 \times 5 = 1,775$ nodes.

To care for 10 L2 sequencers with 2 minutes prove finality, we would need a fleet of $1,775 \times 10 = 17,775$ nodes. This is a crazy large amount of nodes and if successful, it would be an interesting parallel effort.

5.2 An alternative to Risc0's Bonsai

Bonsai is Risc0's remote proving solution that relies on powerful GPUs. Given that it is a centralized parallel proving(no segmentation though) service, we can provide an alternative decentralized solution with parallel proving features.

6 Peer matching via verifiable benchmarks

Noise thrives in p2p networks. Being trustless means that a p2p network has to spend a considerable amount of resources to counter time wasters. This is why we factored in a 1:4 compute redundancy ratio as a precautionary measure in previous sections. To achieve the time standards of centralized services for proving, our p2p network needs to have a mechanism for atomic matching. By atomic we mean that a PoW scheme is employed where prover nodes are required to run it periodically to ensure clients and the protocol that they are capable of getting prove works done. Generally prover nodes claim that they boast certain hardware capabilities. But how can we verify their claims? How can we, for example, ensure that a nodes has 8 GB of memory as claimed? The answer is PoW, but it is unfortunately known to waste resources. On the other hand, if a network has to forget about PoW, then it has to employ crypto-economic measures. The usual measures are requiring nodes to stake certain funds and if they did something wrong, funds will be slashed as punishment. This method is usually accompanied by compiling actions history for nodes. Actions history needs maintenance, pointing to the need for vigilante history checking measures for all nodes. This not only bloats the matching procedure, it also needs new measures every so often because wrong doers find novel ways to mess around. So, we better switch back to PoW as it seems to be the only fill or kill switch for matching in the wild.

We propose an experimental PoW mechanism called *verifiable benchmarking system*, VBS for short, where prover nodes are required to carry on certain calculations frequently, like every 2 hours, and score publicly available points. When bidding for proving, they should show the results of the most recent VBS challenge. Now if the result is valid meaning that is not expired and also verified, the match is done and node gets the job. Targeting memory is crucial here because it is not straightforward to check claims about memory. The plan is to prepare a set of guest programs that target various memory amounts. In Risc0, we can adjust segment limit size and produce segments that target specific memory requirements as shown on Table 1. If a prover node, for example, claims to have 16 GB of memory, we send her a segment that when proved with a lower memory, simply panics and fails to generate a correct proof. To make it easier for the prover nodes on doing wasteful computations, we can even send pending prove jobs as potential benchmark challenges solving two problems: one, reducing wasteful computations, and two, contributing to the network compute capacity.

Conclusion

dStorage provides a great opportunity for parallel proving of Risc0 zkVM programs. By segmenting a large Risc0 guest program into smaller ones whose proving is quick and requires small memory footprint, we can tap into the idle compute potential of Swarm network. Segmenting introduces inflation to the total cycles to be proved but the gains would beat the overhead if a proper protocol with optimum parameters is found.

Parallel proving finds applications in ETH L2 sequencers where the gained speedups in proving accompany the much needed decentralization guarantee of the underlying p2p compute network. This allows to build censorship-resistant L2 sequencers. *Verifiable inference(zkML)* and *general verifiable computing networks* are other application areas where verification plays the pivotal role. Being noticeably large in size, zkVM programs of these niches are in crucial need of parallel proving infrastructure.

Storage-maxi nodes are pretty weak computers on their own but when mobilized effectively, have the potential to challenge heavy duty cloud computers in verifiable computing.

A p2p compute network needs atomic matching to reduce the overall latency. With VBS, a PoW scheme, we can establish robust matching constraints and fight noise more effectively. To reduce the compute waste of PoW, outstanding prove jobs can be used as challenges pushing network efficiency even higher.

Verifiable computing is the end game.

References

[Bruestle et al., 2022] Bruestle, J., Gafni, P., and the RISC Zero Team (2022). *RISC Zero zkVM: scalable, transparent arguments of RISC-V integrity*. https://www.risczero.com/proof-system-in-detail.pdf. Accessed on 2024-07-17.

[Hasanzadeh, 2024] Hasanzadeh, R. (2024). *Risc0 Segmentation Test Results*. https://github.com/WholesumNet/docs/blob/main/parallelization/segmentation/report/report.pdf. Accessed on 2024-07-17.

[The Swarm Team, 2021] The Swarm Team (2021). *Swarm Whitepaper*. https://www.ethswarm.org/swarm-whitepaper.pdf. Accessed on 2024-07-17.