ACM CCS '24 Artifact Appendix: Tight ZK CPU Batched ZK Branching with Cost Proportional to Evaluated Instruction

Yibin Yang
Georgia Institute of Technology, USA
yyang811@gatech.edu

A Artifact Appendix

A.1 Abstract

The artifact includes code for all benchmarks presented in the paper. It mainly includes the interactive ZK scheme in our paper: TightZKCPU. It also includes the baselines.

This document describes how one can use our code to reproduce *all* results in Section 6 of the proceedings paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

GitHub link:
https://github.com/gconeice/tight-vole-zk-cpu
We will maintain new versions, and this appendix will be included and updated accordingly in our repository.

A.2.3 Hardware dependencies

Our repository can be executed on a single machine to emulate ZK Prover P and ZK Verifier V using a localhost network. However, our results were tested using two standalone machines: one for ZK Prover P, and another for ZK Verifier V.

We tested our code on two machines, each having $\geq$ 32GiB memory (except in the case of Figures 9, which include intensive instances). Namely, we used two Amazon Web Services (AWS) EC2 **m5.2xlarge** machines.

For the test case in Figure 9, we tested our baseline Batchman using two machines, each having $\geq$ 100GiB memory. Namely, we used two Amazon Web Services (AWS) EC2 **m5.8xlarge** machines. To fairly compare with our baseline for Figure 9, we also use two **m5.8xlarge** machines. (However, using this powerful machine for Figure 9 is not necessary for our scheme.)

We only tested it over x86_64 CPUs, but we believe ARM CPUs (i.e., Apple M1) should also work.

A.2.4 Software dependencies

We tested our code on a clean installation of Ubuntu 22.04. Our repository includes simple scripts to install everything starting from a clean installation.

We depend on Batchman[1] and the state-of-the-art ZK RAM[2]. They are both developed based on the EMP-toolkit[3] (in particular, the VOLE functionalities inside). Our scripts will help you set it up properly.

A.3 Set-Up

You can simply download our repository and type **"sudo bash setup.sh"**. Just hit 'return/enter' button on the keyboard whenever a question shows.

A.3.1 Installation

You can simply download our repository and type **"bash install.sh"**.

A.3.2 Basic toy test

Note that we have two machines – P and V. For P and V, goto the folder `build`. Let `ip` denote the machine P's IP address. Please set environment variable 'IP=`ip`' on V's machine.

P executes: `./bin/test_pathzk_test 1 12345 localhost`

V executes: `./bin/test_pathzk_test 2 12345 $IP`

If everything goes through, you should see "...`Path Length`..." on V's terminal and execution times on P and V. On the other hand, if something goes wrong, you will see the corresponding error messages. (The experiment can take a few seconds.)

A.3.3 Expected executable files

You should generate the following executable files located in `build/bin/`:

[1] https://github.com/gconeice/stacking-vole-zk
[2] https://github.com/gconeice/improved-zk-ram
[3] https://github.com/emp-toolkit

1. `test_pathzk_comp_batchman_balance`: Tight ZK CPU with balanced instructions and without the rounding optimization.

   Usage: `test_pathzk_comp_batchman_balance PARTY PORT IP #BRANCH #REG #STEP #CIR` where `PARTY`=1,2, `#BRANCH` denotes the number of instructions, `#REG` denotes the number of register, `#STEP` denotes the number of steps to execute, and `#CIR` denotes the size of each instruction.

2. `test_pathzk_comp_batchman_balance_opt`: Tight ZK CPU with balanced instructions and the rounding optimization.

   Usage: `test_pathzk_comp_batchman_balance_opt PARTY PORT IP #BRANCH #REG #STEP #CIR` where `PARTY`=1,2, `#BRANCH` denotes the number of instructions, `#REG` denotes the number of register, `#STEP` denotes the number of steps to execute, and `#CIR` denotes the size of each instruction.

3. `test_pathzk_comp_batchman_unbalance`: Tight ZK CPU with unbalanced instructions and without the rounding optimization.

   Usage: `test_pathzk_comp_batchman_unbalance PARTY PORT IP #BRANCH #REG #STEP #CIR` where `PARTY`=1,2, `#BRANCH` denotes the number of instructions, `#REG` denotes the number of register, `#STEP` denotes the number of steps to execute, and `#CIR` denotes the size of (the largest) instruction.

4. `test_pathzk_comp_batchman_uniform`: Tight ZK CPU with uniformly distributed sizes of instructions and without the rounding optimization.

   Usage: `test_pathzk_comp_batchman_uniform PARTY PORT IP #BRANCH #REG #STEP #CIR` where `PARTY`=1,2, `#BRANCH` denotes the number of instructions, `#REG` denotes the number of register, `#STEP` denotes the number of steps to execute, and `#CIR` denotes the size growing factor of each instruction. (E.g., when `#CIR`= 10, the instructions would be of size $10, 20, 30, \ldots$.)

5. `test_pathzk_comp_with_pub_cir`: Tight ZK CPU with uniformly distributed sizes of instructions and without the rounding optimization, with inlined parameters to compare with the *baseline insecure execution*. (The execution would generate files that can be re-executed by the *baseline insecure execution* with the same CPU configuration.)

   Usage: `test_pathzk_comp_with_pub_cir PARTY PORT IP` where `PARTY`=1,2.

6. `test_pathzk_comp_with_pub_cir_opt`: Tight ZK CPU with uniformly distributed sizes of instructions and the rounding optimization, with inlined parameters to compare with the *baseline insecure execution*. (The execution would generate files that can be re-executed by the *baseline insecure execution* with the same CPU configuration.)

   Usage: `test_pathzk_comp_with_pub_cir_opt PARTY PORT IP` where `PARTY`=1,2.

7. `test_pathzk_pub_cir`: The baseline of the *insecure execution*.

   Usage: `test_pathzk_pub_cir PARTY PORT IP CPU_CFG_FILE INPUT_FILE FINAL_REG_FILE CID_FILE` where `PARTY`=1,2.

8. `test_arith_stack_batched_matmul_v1`: The baseline Batchman on $\mathbb{F}_{2^{61}-1}$ for the matrix multiplications.

   Usage: `test_arith_stack_batched_matmul_v1 PARTY PORT IP LEN #BRANCH #REPETITION`.

9. `test_pathzk_fine_grain`: The fine-grained analysis in the appendix H.

   Usage: `test_pathzk_fine_grain PARTY PORT IP #BRANCH #REG #STEP #CIR` where `PARTY`=1,2, `#BRANCH` denotes the number of instructions, `#REG` denotes the number of register, `#STEP` denotes the number of steps to execute, and `#CIR` denotes the size of each instruction.

10. `test_pathzk_test`: A clean version, which is mainly used for the toy example.

A.4 Evaluation Workflow

Please set environment variable 'IP=`ip`' on V's machine, where `ip` is the machine P's IP address.

A.4.1 Major Claims

(C1): The performance of our VOLE-based tight ZK CPU is illustrated/reported in Figure 7. This is proven by the experiment (E1) described in Section 6.

(C2): The performance of our VOLE-based tight ZK CPU, compared to the baseline Batchman, is illustrated/reported in Figures 8 and 9. This is proven by the experiment (E2/E3) described in Section 6.

(C3): The performance of our VOLE-based tight ZK CPU, compared to the *insecure execution* baseline, is illustrated/reported in Figure 10. This is proven by the experiment (E4) described in Section 6.

(C4): The microbenchmarks of our VOLE-based tight ZK CPU are illustrated/reported in Figures 14 and 15. This is proven by the experiment (E5) described in Appendix H.

A.4.2 Experiments

Note: All our figures/tables are plotted based on the data in the Excel file `benchmark_summary.xlsx`. Therefore, we will show how one can reproduce the numbers in this Excel file and how to transform them into figures/tables.

(E1): [Figure 7] [10 human-minutes + 20 compute-minutes of two machines + 32GB memory each machine/party]: Please `cd` to the folder `build`.

Machines we used: AWS EC2 m5.2xlarge.

Preparation: For both machines, let the name of the network card be `ens5`; please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 100Mbit burst 100000 limit 10000` (resp. `500Mbit, 1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is either 100/500 Mbps or 1 Gbps. You can use `iperf` to check it.

Data in Excel: Please refer to the sheet `Figure 7`.

Execution: We need to perform the following experiments over balanced/unbalanced/varied distribution. For each network setting, the following execution needs to be executed repeatedly (i.e., 3 times).

- Balanced:
  For each $(\#BRANCH, \#REG, \#STEP, \#CIR) = (10,5,50000,100), (50,1,50000,100), (50,10,50000,100), (50,20,50000,100), (100,20,50000,100)$:
  P machine:
  ```
  ./bin/test_pathzk_comp_batchman_balance
  1 12345 localhost #BRANCH #REG #STEP
  #CIR
  ```
  V machine:
  ```
  ./bin/test_pathzk_comp_batchman_balance
  2 12345 $IP #BRANCH #REG #STEP #CIR
  ```
- Unbalanced:
  For $(\#BRANCH, \#REG, \#STEP, \#CIR) = (100,20,50000,100)$:
  P machine:
  ```
  ./bin/test_pathzk_comp_batchman_unbalance
  1 12345 localhost #BRANCH #REG #STEP
  #CIR
  ```
  V machine:
  ```
  ./bin/test_pathzk_comp_batchman_unbalance
  2 12345 $IP #BRANCH #REG #STEP #CIR
  ```
- Varied:
  For $(\#BRANCH, \#REG, \#STEP, \#CIR) = (100,20,50000,10)$:
  P machine:
  ```
  ./bin/test_pathzk_comp_batchman_uniform
  1 12345 localhost #BRANCH #REG #STEP
  #CIR
  ```
  V machine:
  ```
  ./bin/test_pathzk_comp_batchman_uniform
  2 12345 $IP #BRANCH #REG #STEP #CIR
  ```

Results: The time outputted on P and V's terminals reflects the number in Figure 7 (Excel and the paper). In particular, the `path` column in the Excel reflects the `path length` on the terminal (P and V should be the same); the `time` column in the Excel reflects the `xxx us 2` in the V's terminal (note that it is in microsecond in the terminal while it is second in the Excel); the `P` $\rightarrow$ `V` column in the Excel reflects the `communication` in P's terminal (in Byte); the `V` $\rightarrow$ `P` column in the Excel reflects the `communication` in V's terminal (in Byte).

Compute MGPS and CPM: MGPS is computed by `Path`/`Time` in the Excel; CPM is computed by $((\text{P}\rightarrow\text{V})+(\text{V}\rightarrow\text{P}))/$`Path` in the Excel.

Nondeterminism: We remark that for the unbalanced and varied test cases, since the execution path is randomized, the numbers can be slightly deviated from what is in the Excel. However, the MGPS and CPM would not differ a lot.

On the other hand, for the balanced test cases, all numbers should meet the numbers in Excel since there is no random path selection.

(E2): [Figure 8] [10 human-minutes + 20 compute-hours of two machines + 32GB memory each machine/party]: Please `cd` to the folder `build`.

Machines we used: AWS EC2 m5.2xlarge.

Preparation: For both machines, let the name of the network card be `ens5`; please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 100Mbit burst 100000 limit 10000` (resp. `500Mbit, 1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is either 100/500 Mbps or 1 Gbps. You can use `iperf` to check it.

Data in Excel: Please refer to the sheet `Figure 8 9` and focus on the `Figure 8` part.

Execution: We need to perform the following experiments over balanced/unbalanced/balanced(with opt) distribution as well as the baseline Batchman. For each network setting, the following execution needs to be executed repeatedly (i.e., 3 times).

- Baseline Batchman:
  For $(LEN, \#BRANCH, \#REPETITION) =$

$(5,50,500000)$
P machine:
```
./bin/test_arith_stack_batched_matmul_v1
1 12345 localhost LEN #BRANCH
#REPETITION
```
V machine:
```
./bin/test_arith_stack_batched_matmul_v1
2 12345 $IP LEN #BRANCH #REPETITION
```

- Balanced: For $(\#BRANCH, \#REG, \#STEP, \#CIR) = (50,1,500000,125)$:
  P machine:
  ```
  ./bin/test_pathzk_comp_batchman_balance
  1 12345 localhost #BRANCH #REG #STEP
  #CIR
  ```
  V machine:
  ```
  ./bin/test_pathzk_comp_batchman_balance
  2 12345 $IP #BRANCH #REG #STEP #CIR
  ```
- Unbalanced:
  For $(\#BRANCH, \#REG, \#STEP, \#CIR) = (50,1,500000,125)$:
  P machine:
  ```
  ./bin/test_pathzk_comp_batchman_unbalance
  1 12345 localhost #BRANCH #REG #STEP
  #CIR
  ```
  V machine:
  ```
  ./bin/test_pathzk_comp_batchman_unbalance
  2 12345 $IP #BRANCH #REG #STEP #CIR
  ```
- Balanced, Opt:
  For $(\#BRANCH, \#REG, \#STEP, \#CIR) = (50,1,500000,125)$:
  P machine:
  ```
  ./bin/test_pathzk_comp_batchman_balance_opt
  1 12345 localhost #BRANCH #REG #STEP
  #CIR
  ```
  V machine:
  ```
  ./bin/test_pathzk_comp_batchman_balance_opt
  2 12345 $IP #BRANCH #REG #STEP #CIR
  ```

Results: The time outputted on P and V's terminals reflects the number in Figure 8 (Excel and the paper). In particular, the `time` cells (i.e., those w/o annotating) in the Excel reflect the `xxx us 2` in the V's terminal (note that it is in microsecond in the terminal while it is second in the Excel); the `P` $\rightarrow$ `V` column in the Excel reflects the `communication` in P's terminal (in Byte); the `V` $\rightarrow$ `P` column in the Excel reflects the `communication` in V's terminal (in Byte).

Compute Hz Rate and Comm./Step.: Note that all the experiments are configured with 500000 steps. The Hz Rate can be computed via $500000/$`Time`; and the Comm./Step. can be computed via $((\text{P}\rightarrow\text{V}) + (\text{V}\rightarrow\text{P}))/500000$ in the Excel. The speedup is defined by the ratio of ours/baseline.

Nondeterminism: We remark that for the unbalanced test cases, since the execution path is randomized, the numbers may be slightly deviated from what is in the Excel. However, the Hz Rate and Comm./Step. would not differ a lot.
On the other hand, for the balanced test cases (opt or w/o opt) and the baseline, all numbers should meet the numbers in Excel since there is no random path selection.

(E3): [Figure 9] [10 human-minutes + 30 compute-hours of two machines + 128GB memory each machine/party]: Please `cd` to the folder `build`. **We remark that** E3 is essentially same as E2 with larger test cases. To support our baseline, we need a stronger machine with a larger RAM.
Machines we used: AWS EC2 m5.8xlarge.
Preparation: For both machines, let the name of the network card be `ens5`; please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 100Mbit burst 100000 limit 10000` (resp. `500Mbit`, `1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is either $100/500$ Mbps or 1 Gbps. You can use `iperf` to check it.
Data in Excel: Please refer to the sheet `Figure 8 9` and focus on the `Figure 9` part.
Execution: We need to perform the following experiments over unbalanced distribution and the baseline Batchman. For each network setting, the following execution needs to be executed repeatedly (i.e., 3 times).

- Baseline Batchman:
  For $(LEN, \#BRANCH, \#REPETITION) = (10,50,500000)$
  P machine:
  ```
  ./bin/test_arith_stack_batched_matmul_v1
  1 12345 localhost LEN #BRANCH
  #REPETITION
  ```
  V machine:
  ```
  ./bin/test_arith_stack_batched_matmul_v1
  2 12345 $IP LEN #BRANCH #REPETITION
  ```
- Unbalanced:
  For $(\#BRANCH, \#REG, \#STEP, \#CIR) = (50,1,500000,1000)$:
  P machine:
  ```
  ./bin/test_pathzk_comp_batchman_unbalance
  1 12345 localhost #BRANCH #REG #STEP
  #CIR
  ```
  V machine:
  ```
  ./bin/test_pathzk_comp_batchman_unbalance
  2 12345 $IP #BRANCH #REG #STEP #CIR
  ```

Results: The time outputted on P and V's terminals

reflects the number in Figure 9 (Excel and the paper). In particular, the `time` cells (i.e., those w/o annotating) in the Excel reflect the `xxx us 2` in the V's terminal (note that it is in microsecond in the terminal while it is second in the Excel); the `P → V` column in the Excel reflects the `communication` in P's terminal (in Byte); the `V → P` column in the Excel reflects the `communication` in V's terminal (in Byte).

Compute Hz Rate and Comm./Step.: Note that all the experiments are configured with 500000 steps. The Hz Rate can be computed via 500000/`Time`; and the Comm./Step. can be computed via $((\text{P}\rightarrow\text{V}) + (\text{V}\rightarrow\text{P}))/500000$ in the Excel. The speedup is defined by the ratio of ours/baseline.

Nondeterminism: We remark that for the unbalanced test cases, since the execution path is randomized, the numbers may be slightly deviated from what is in the Excel. However, the Hz Rate and Comm./Step. would not differ a lot.

On the other hand, for the baseline, all numbers should meet the numbers in Excel since there is no random path selection.

(E4): [Figure 10] [10 human-minutes + 30 compute-hours of two machines + 32GB memory each machine/party]: Please `cd` to the folder `build`.

Machines we used: AWS EC2 m5.2xlarge.

Preparation: For both machines, let the name of the network card be `ens5`; please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 100Mbit burst 100000 limit 10000` (resp. `500Mbit`, `1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is either 100/500 Mbps or 1 Gbps. You can use `iperf` to check it.

Data in Excel: Please refer to the sheet `Figure 10`.

Execution: We need to perform the following experiments over varied distribution and the baseline QuickSilver. For each network setting, the following execution needs to be executed repeatedly (i.e., 3 times). Note that the baseline needs to read files generated by our protocol. Thus, please test our protocol first and then the baseline.

- Varied:
  P machine:
  ```
  ./bin/test_pathzk_comp_with_pub_cir 1 12345 localhost
  ```
  V machine:
  ```
  ./bin/test_pathzk_comp_with_pub_cir 2 12345 $IP
  ```
  This will also generate files: `cpu.cfg`, `input.txt`, `finalreg.txt`, and `cid.txt` on P's machine. Please move `input.txt`, `finalreg.txt`, and `cid.txt` to V's machine under the `build` folder. E.g., you can use `scp` to do so by downloading and then uploading.
- Baseline QuickSilver:
  P machine:
  ```
  ./bin/test_pathzk_pub_cir 1 12345 localhost cpu.cfg input.txt finalreg.txt cid.txt
  ```
  V machine:
  ```
  ./bin/test_pathzk_pub_cir 2 12345 $IP cpu.cfg input.txt finalreg.txt cid.txt
  ```

Results: The time outputted on P and V's terminals reflects the number in Figure 10 (Excel and the paper). In particular, the `time` cells (i.e., those w/o annotating) in the Excel reflect the `xxx us 2` in the V's terminal (note that it is in microsecond in the terminal while it is second in the Excel); the `P → V` column in the Excel reflects the `communication` in P's terminal (in Byte); the `V → P` column in the Excel reflects the `communication` in V's terminal (in Byte). (Note: The `Size` Column in the Excel, while not important, reflects the first number outputted by the QuickSilver experiments.)

Compute Speedup: The factor blowup in Figure 10 is defined by the ratio of ours/baseline.

Nondeterminism: We remark that for the varied test cases (i.e., ours), since the execution path is randomized, the numbers may be slightly deviated from what is in the Excel. However, the factor blowup should be similar.

(E5): [Figure 14, 15] [10 human-minutes + 30 compute-hours of two machines + 32GB memory each machine/party]: Please `cd` to the folder `build`.

Machines we used: AWS EC2 m5.2xlarge.

Preparation: For both machines, let the name of the network card be `ens5`; please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 100Mbit burst 100000 limit 10000` (resp. `500Mbit`, `1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is either 100/500 Mbps or 1 Gbps. You can use `iperf` to check it.

Data in Excel: Please refer to the sheet `fine grain`.

Execution: We need to perform the following experiments over varied distribution. For each network setting, the following execution needs to be executed repeatedly

(i.e., 3 times).

- Varied:
  P machine:
  ```
  ./bin/test_pathzk_fine_grain 1 12345
  localhost 50 20 50000 10
  ```
  V machine:
  ```
  ./bin/test_pathzk_fine_grain 2 12345
  $IP 50 20 50000 10
  ```

Results: The time outputted on P and V's terminals reflects the number in Figures 14 and 15 (Excel and the paper). In particular, the `time` cells in the Excel reflect the `xxx us 2` in the V's terminal (note that it is in microsecond in the terminal while it is second in the Excel). The time tests are outputted in order (cf. Figures 14 and 15).

Typo: We note that there is a subtle typo in the paper where the caption of Figures 14 and 15 should be with 50K steps rather than 500K steps. We will fix this typo in the final version.

Nondeterminism: We remark that for the varied test cases, since the execution path is randomized, the numbers may be slightly deviated from what is in the Excel.