Verifiable Computation with Massively Parallel Interactive Proofs

Justin Thaler, Harvard University

Mike Roberts, Harvard University

Michael Mitzenmacher, Harvard University

Hanspeter Pfister, Harvard University

Outsourcing

- Many applications require outsourcing computation to untrusted service providers.
  - Main motivation: Commercial cloud computing services.
  - Also, weak peripheral devices; fast but faulty co-processors.
  - Volunteer Computing (SETI@home, World Community Grid, etc.)
- User requires a guarantee that the cloud performed the computation correctly.
- One solution: require cloud to *prove* correctness of answer.

Goals of Verifiable Computation

- Provide user with a correctness guarantee, without requiring her to perform the requested computations herself.
  - Ideally user will not even maintain a local copy of the data.
  - User may have resorted to the cloud in the first place because she has more data than she can store.

- Minimize the amount of extra bookkeeping the cloud has to do to prove the integrity of the computation.

- Ideally our protocols will be secure against arbitrarily malicious clouds, but sufficiently lightweight for use in more benign settings.

Interactive Proofs

- Two Parties: Prover P and Verifier V.
- Think of P and powerful, V as weak. P solves a problem, tells V the answer.
  - Then P and V have a conversation.
  - P's goal: convince V the answer is correct.
- Requirements:
  - 1. Completeness: An honest P can convince V she's telling the truth.
  - 2. Soundness: V will catch a lying P with high probability no matter what P says to try to convince V (Secure even if P is computationally unbounded).

Interactive Proofs

- IPs have revolutionized Complexity Theory in the last 25 years.
  - IP=PSPACE [Shamir 90].
  - PCP Theorem e.g. [AS 98]. Hardness of approximation.
  - Zero Knowledge Proofs.

- But IPs have had very little impact in real delegation scenarios.
  - Why?
  - Not due to lack of applications!

Interactive Proofs

- Old Answer: Most results on IPs dealt with hard problems, needed P to be too powerful.
  - But recent constructions focus on "easy" problems (e.g. "Interactive Proofs for Muggles" [GKR 08]).
  - Allows V to run **very** quickly, so outsourcing is useful even though problems are "easy".
  - P does not need "much" more time to prove correctness than she does to solve the problem in the first place!

Interactive Proofs

- Why does GKR not yield a practical protocol out of the box?
  - P has to do a lot of extra bookkeeping (**cubic** blowup in runtime).
  - Naively, V has to retain the full input.
  - Substantial overhead due to finite field arithmetic and other technical issues.

Engineering Practical IPs [CMT12, TRMP12]

A Two-Pronged Approach

- The present paper is part of a recent line of work aiming to develop practical IPs [CCMT12, CMT10, CTY12, CMT12]
- Ideal: General purpose implementation allowing to verify arbitrary computation.
  - Based on general-purpose "Interactive Proofs for Muggles" construction [GKR 08].
- Also develop highly optimized protocols for specific important problems.
  - Reporting queries (what value is stored in memory location x of my database?)
  - Matrix multiplication.
  - Graph problems like perfect matching.
  - Certain kinds of linear programs.
  - Etc.

Main Results: Part 1

- Can save V substantial amounts of space essentially for free.
  - Reason: GKR protocol (and several others) only requires V to store a fingerprint of the data.
  - This fingerprint can be computed in a single, light-weight pass over the input.
  - Fingerprint serves as a sort of "secret" that V can use to catch the cloud in a lie.
- Fits cloud computing well: pass by V can occur while uploading data to cloud.
- V never needs to store entirety of data!
- The fingerprint is a few KBs in size, even if the input contains terabytes of data.

Main Results: Part 2

- Can save V substantial amounts of time.
- E.g. when multiplying two 512x512 matrices, V requires .12s to process the input, while naive matrix multiplication takes about .70 seconds.
- Savings for V will be much larger on at larger input sizes, when applying our implementation to more time-intensive computations than matrix multiplication (because V's runtime grows quasi-linearly with input size; she just needs to compute a fingerprint of the input).

Main Results: Part 3

- We've come a long way in making P more efficient.
- In [CMT12], we brought the runtime of P down from **cubic** in the size of a circuit computing the function of interest, to quasilinear in the size of the circuit.
- Lots of additional engineering in the implementation (helps make V fast too).
  - Choosing the “right” finite field to work over.
  - Using the “right” circuits.
  - Etc.
- Practically speaking, this is still not good enough on its own.
  - 256 x 256 matrix multiplication takes P about 27 minutes for our previous single-threaded implementation.

Main Results: Part 4 (Focus of [TRMP12])

- Our implementation is extremely amenable to parallelization.
- Holds for both P and V (although V runs quickly even without parallelization, see Insight 2).

Problem	P time (single-threaded)	P time (GPU)	V time	Rounds	Communication
F_2 (n=2^20)	29.8 s	0.36 s	.19 s	118	2.5 KB
MatMult (256 x 256)	27.6 Minutes	39.6.s	.04 s	3910	91.6 KB

If V also has a GPU, we get close to 100-fold speedups for V relative to single-threaded implementation.

Main Results: Part 4 (Focus of [TRMP12])

- Main challenge to parallelizing and scaling to large inputs was the memory-intensive nature of P's computation in the GKR protocol.
  - Naïve n x n matrix multiplication only requires $O(n^2)$ space.
  - P has to store a circuit of size $O(n^3)$ (we use 40 bytes per gate).
  - Even 256 x 256 matrix multiplication over 1.5 GBs of space.
  - Took steps to mitigate this issue despite limited device memory.

Related Work

- Setty, McPherson, Blumberg, and Walfish [NDSS 12] implement an *argument* system original due to Ishai, Kushilevitz, and Ostrovsky [CCC 07].
  - Bring the runtime of the cloud down by a factor of 10^20 relative to a naive implementation.
  - Advantages of our implementation: save V time even when outsourcing a single computation, secure against computationally unbounded clouds.
- Canetti, Riva, and Rothblum [CCS 12] give highly practical protocols which are secure when there are *two* clouds, at least one of whom is honest.
- Ben-Sasson, Chiesa, Genkin, and Tromer working toward practical PCPs.

Conclusions

- Interactive Proofs and other protocols for verifiable computation represent some of the most celebrated results in complexity theory.
- They have the potential to mitigate trust issues in cloud computing, but were wildly impractical until recently.
- We can already save the user a lot of time and space.
- The main remaining bottleneck is the extra bookkeeping the cloud must do to provide integrity guarantees.
- Parallelization helps mitigate this issue, but there is still much work to be done.

Thank you!

Sample Variance of Data Stream

- The (scaled) sample variance of a data stream is defined as follows:
  - Let **X** be the frequency vector of the stream
    ($\mathbf{X}_i$ is number of occurrences of i in the stream)
  - $F_2(\mathbf{X})=\sum_i \mathbf{X}_i^2$

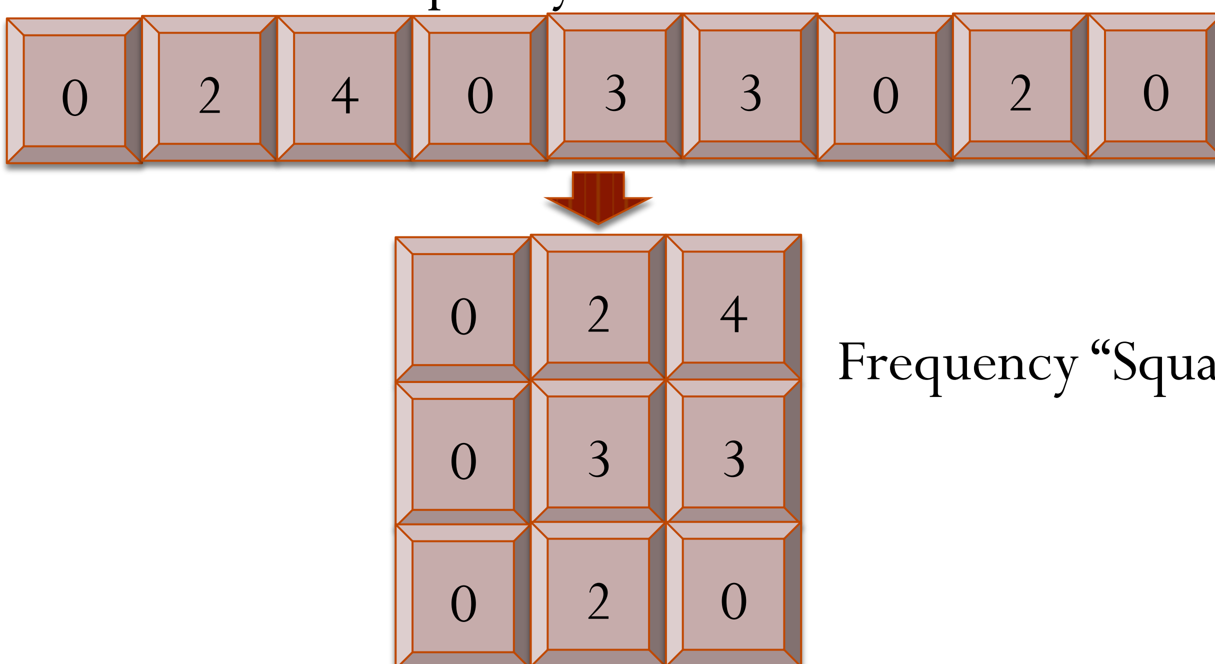
- [CCM 09/CCMT 12] give a one-message protocol for F_2 requiring $O(\sqrt{n})$ communication and $O(\sqrt{n})$ space for V.

- This is optimal.

Sample-Variance Protocol

- Recall: $F_2(\mathbf{X}) = \sum_i \mathbf{X}_i^2$
- View universe [n] as [√n] x [√n].

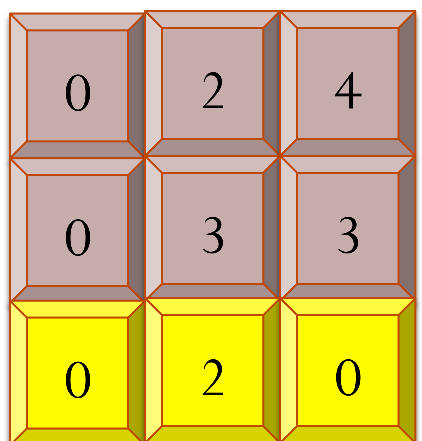
Frequency Vector **X**



Frequency “Square” **X**

- First idea: Have P send the answer "in pieces":
  - F_2(row 1). F_2(row 2). And so on. Requires $\sqrt{n}$ communication.
- V exactly tracks a row at random (denoted in yellow) so if P lies about any piece, V has a chance of catching her. Requires space $\sqrt{n}$.

Frequency Square **X**



P sends

$20=2^2+4^2$

$18=3^2+3^2$

$4=2^2$

Slide derived from [McGregor 10]

- Problem: If P lies in only one place, V has small chance of catching her.
- We would like the following to hold: if P lies about even one piece, she will have to lie about many.
- Solution: Have P commit (succinctly) to second frequency moment of rows of an **error-corrected encoding** of the input.
- Need V to evaluate any row of the encoding in a streaming fashion. Can do this for "low-degree extension" code. Note: this code is *systematic*, meaning the first n symbols are just the input itself.

These values will all lie on low-degree polynomial s(X)

Error-corrected Encoding of Frequency Square X

Input is embedded in encoding (low-degree extension)

0	2	4
0	3	3
0	2	0
0	-1	-5
0	-6	-12
0	-13	-21

H sends

$20=2^2+4^2$

$18=3^2+3^2$

$4=2^2$

$26=(-1)^2+(-5)^2$

$180=(-6)^2+(-12)^2$

$610=(-13)^2+(-21)^2$