Practical Verified Computation with Streaming Interactive Proofs

Justin Thaler, Harvard University

Joint work with:

Graham Cormode (AT&T Labs – Research),

Michael Mitzenmacher (Harvard)

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 service provider performed the computation correctly.

- One solution: require provider to *prove* correctness of answer.

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.

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]).
  - Allow V to run **very** quickly, use small space, 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.

- Shouldn’t these results be useful and exciting to practitioners?

New Application of IPs

- To streaming problems: hard because V has to read input in one-pass streaming manner, but (might be) easy if V could store the whole input. [CCM 09/CCMT 12], [CMT 10], [CTY 11].

- Fits cloud computing well: streaming pass by V can occur while uploading data to cloud.

- V never needs to store entirety of data.

- [GKR 08] actually works with streaming verifier. [CCM 09/CCMT 12], [CMT 10], [CTY 10] give improved protocols for large classes of important problems (LPs, graph problems, frequency moments, etc).

This Work: A Two-Pronged Approach

- Ideal: General purpose implementation allowing to verify arbitrary computation.
  - We revisit general-purpose "Interactive Proofs for Muggles" construction of [GKR 08], with focus on speeding up prover.
  - Plus extensions to help move it from theory to practice.
  - Full implementation, with encouraging experimental results.

- Also revisit specialized protocols of [CCM 09/CCMT 12], [CMT 10], [CTY 11].
  - Speed up prover for these too.
  - Full implementation, with encouraging experimental results.

First Prong: General Purpose Construction

Speeding up P in "Muggles" [GKR08]

- In "Muggles", P and V first agree on a layered arithmetic circuit C computing the function of interest. Then P gives V the output of C and proves that the output is correct.
  - Runs a *sum-check protocol* for each layer in turn, in order to check that all outputs of each layer are computed correctly.
- A naïve implementation of P would require $\Omega(S^3)$ time, where S is size of circuit C, because each message P sends involves a sum over S^3 terms.
- We engineer P's time down to O(S log S), under mild conditions on the circuit.

A general technique

- Arithmetization: Given function *f'* defined on a small domain, extend domain of *f'* to a large field and replace *f'* with its low-degree extension (LDE) f as a polynomial over the field.

- Can view *f* as an error-corrected encoding of *f'*. The error correcting properties of *f* give V considerable power over P.

- If two functions differ in one location, their LDE's will differ in almost all locations.

Speeding up P in "Muggles" [GKR08]

- The ith sum-check protocol makes use of the following functions.
  - Let v = log n. jth gate at layer i is associated with v-bit representation of j.
  - Functions add-i, mult-i (3v bits to 1 bit)
    - add-i(a,b,c) = 1 if gate a at layer i is sum of gates b, c from layer i-1
    - Extend to multilinear low-degree extension (this is a **SPECIAL** low-degree extension)

- Key point: If you use the **multilinear extension** of add-i and mult-i, the sum defining P's message simplifies nicely.
  - Each gate in layer i only contributes to a single term in the sum, and this contribution can be computed in O(1) time per gate.
  - So each message can be computed with a single pass over the gates at layer i. So O(S log S) time is required over all rounds.

Speeding up P in "Muggles" [GKR08]

- Complication: V needs to evaluate the multilinear extensions at a random point (rest of V's computation is extremely light-weight).
  - V can do this in log space for *any* log-space uniform circuit. Sufficient for many streaming applications.
  - Moreover, this computation depends only on the circuit, not the input, so it can occur in an *offline* pre-processing phase. V will be both space- and time-efficient in online phase.

Speeding up P in "Muggles" [GKR08]

- No offline phase needed if multilinear extensions of wiring predicate can be evaluated quickly.
- This holds for a class of "reasonably regular" circuits, including:
  1. FFT
  2. Frequency moments
  3. Pattern matching
  4. Matrix multiplication
  5. Circuits from [GKR08] used for simulating space-bounded Turing Machines.

Protocol Engineering: More Types of Gates

- Also extended [GKR08] to allow for more general kinds of gates in C.
  - In particular, "exponentiation" and "sum with big fan-in" gates – reduces the depth of the computation.
    - Another use: computing x^{p-1} mod p gives 1 when x is not zero mod p and 0 otherwise by Fermat's Little Theorem. This is a common primitive e.g. it lets you test vectors for equality.
    - Many protocols terminate by computing a big sum of values, even just once at the end.
  - Smaller depth means fewer rounds of communication, but larger communication size per round. Optimize for the sweet spot in practice.

Protocol Engineering: The Right Field

- We work over F_p for $p = 2^{61} - 1$.
- Keeps values in a single 64-bit data type.
- Miniscule probability of error for practical purposes.
  - Errors proportional to $1/p$.
- Reducing modulo p can be done with a bit shift and a bit-wise AND operation.
  - About an order of magnitude faster than standard mod operation.

“Muggles” Implementation

Problem	Gates	Size (gates)	P time	V time	Rounds	Comm
F_2	$+, \times$	0.4M	8.5 s	.01 s	986	11.5 KB
F_2	$+, \times, \oplus$	0.2M	6.5 s	.01 s	118	2.5 KB
F_0	$+, \times$	16M	552.6 s	.01 s	3730	87.4 KB
F_0	$+, \times, \wedge 8, \oplus$	8.2M	432.6 s	.01 s	1310	51.0 KB
F_0	$+, \times, \wedge 16, \oplus$	6.2M	441.2 s	.01 s	1024	56.8 KB
PMwW	$+, \times, \wedge 8, \oplus$	9.6M	482.2 s	.01 s	1513	56.1 KB

Experimental results with general-purpose implementation, when run on streams with universe size $2^{17}=131{,}072$.

Second Prong: Specialized Protocols

Protocol Engineering: Smart FFTs

- [CCM 09/CCMT 12], [CMT 11] develop *non-interactive* protocols achieving *optimal* tradeoffs between proof length and V's space for a wide variety of streaming problems.
  - Practical: P can send proof via email, or post it on a website for V to retrieve at her convenience.
- Fundamental building block in most of these protocols is an optimal protocol for the *second frequency moment*, or F_2, problem from [CCM 09/CCMT 12].

Protocol Engineering: Smart FFTs

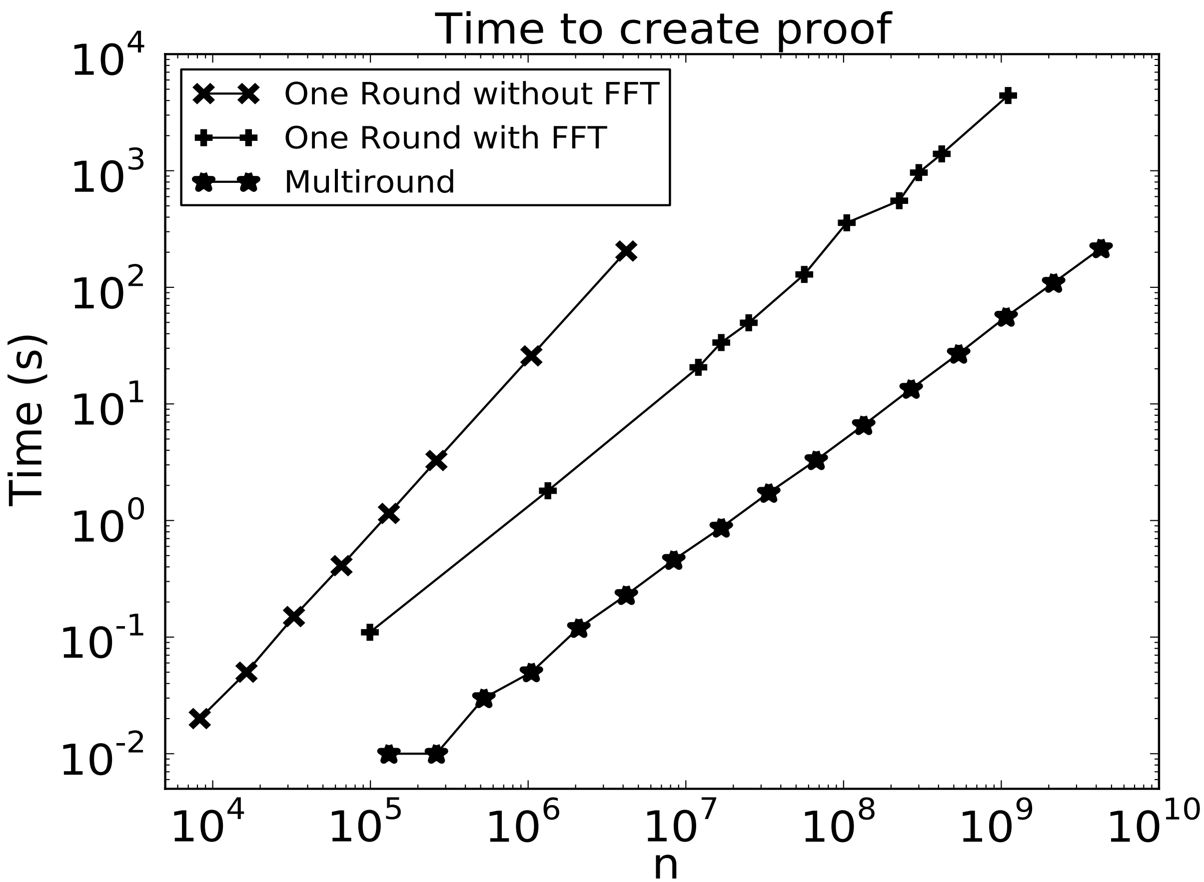
- The F_2 protocol requires P to evaluate the low-degree extension of the input at many points.
  - Naively computing each point independently requires $\Omega(n^{3/2})$ time, doesn't scale to large streams.
  - Using FFT techniques, we can reduce this to O(n log n) time.
- Cooley-Tukey DFT algorithm lets us do this over the complex numbers, with the right amount of bit precision.
  - But this is slow, requires care in handling precision issues.
  - Well over 64 bits precision needed.
  - Only works for power-of-two sized inputs; a lot of padding may be needed.

Protocol Engineering: Smart FFTs

- Used Prime Factor Algorithm instead. This works well over certain finite fields F_p ($p=2^{61}-1$ in particular) and allows us to avoid precision issues entirely.

- Achieved an implementation of P processing 250,000-750,000 updates per second across all stream lengths.

F_2 Experiments



Multi-round P from [CTY11] vs. Non-interactive P with and without FFT techniques

Matrix-Vector Multiplication Experiments

α	**V Space**	**Comm**	**V Time**	**P Time**
1	78 KB	78 KB	4.3 s	1.6 s
.85	20 KB	469 KB	3.0 s	33.9 s
.80	12 KB	938 KB	2.8 s	58.9 s
.75	8 KB	1.5 MB	2.6 s	61.0 s

Results for single-round matrix-vector multiplication protocol from [CMT10] with FFT techniques on 10,000 x 10,000 matrix (768 MBs of data).

Follow-up Work [TRMP 12]

• Both P and V's computations in our implementation are massively parallel.

• Implemented on GPU (massively parallel hardware) and obtained robust speedups (40x-100x for P, 100x for V) relative to the sequential implementation just described.

• Parallel P is roughly 100-1000 times slower than *unverifiable* sequential algorithms for benchmark problems like matrix multiplication and pattern matching.

• For a superlinear-time problem like matrix multiplication, even sequential V is 100s of times faster than doing computation locally.

•Also achieve 50x speedups for special-purpose protocols using GPU processing, relative to sequential implementation.

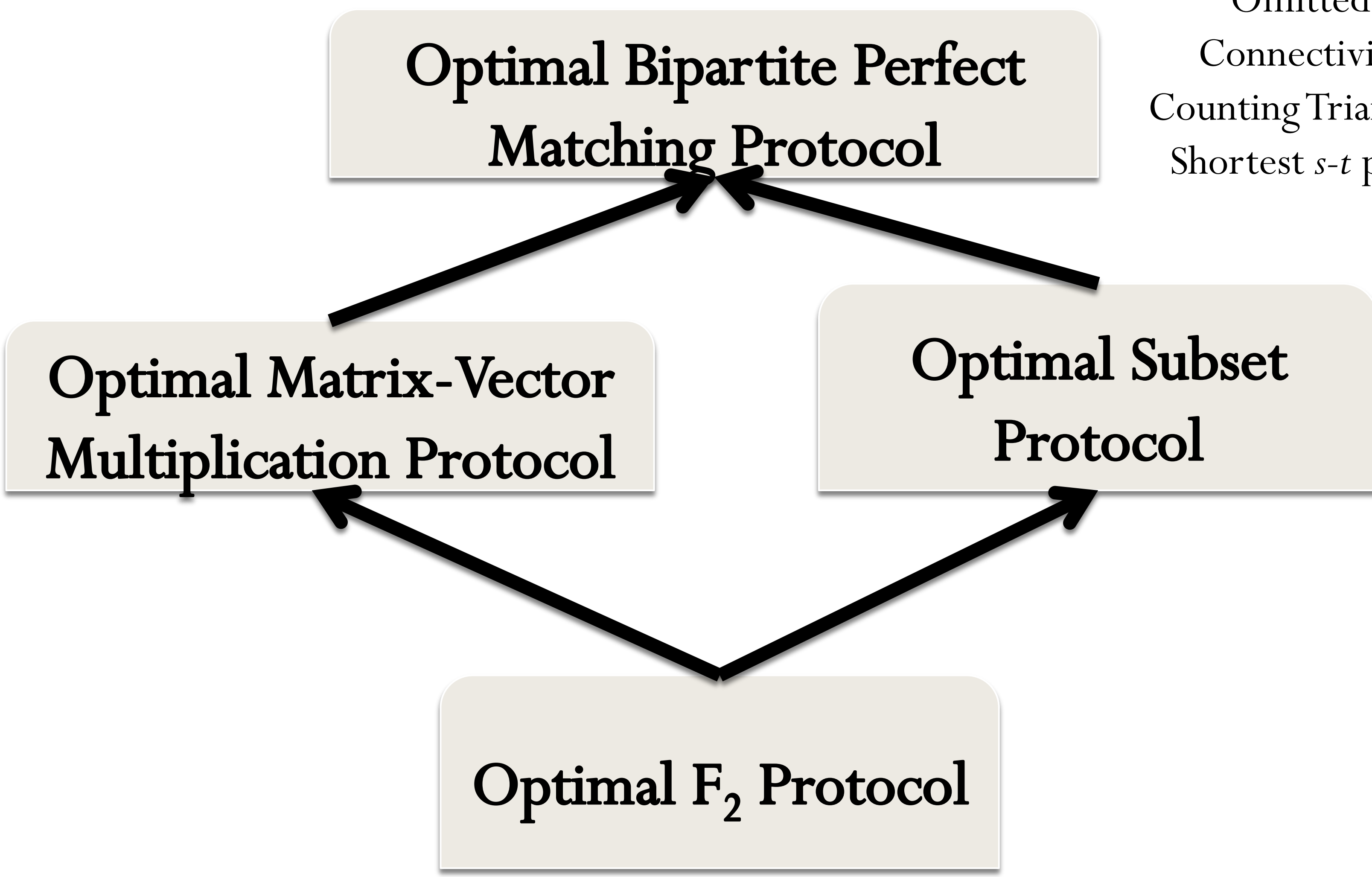
Open Questions

- Non-interactive protocols:
  1. Proving any specific problem requires $\max(h, v) > \sqrt{n}$ requires novel communication complexity techniques.
  2. Improved protocols for sparse graphs?

- Interactive/General Purpose protocols:
  1. Ultimate goal is a general-purpose compiler that takes as input any computer program and outputs a specification of a protocol for running the program *verifiably*.
     - Existing compilers (e.g. Fairplay) construct a *boolean* circuit operating on individual bits, and arithmetize it to get an arithmetic one. But this will lead to impractically large circuits, even for "algebraic" problems which possess small arithmetic circuits.
  2. Improved protocols for "non-algebraic" problems, which may have no small arithmetic circuits.

Thank you!

Why study F_2 ?



Omitted:
Connectivity,
Counting Triangles,
Shortest *s*-*t* path.