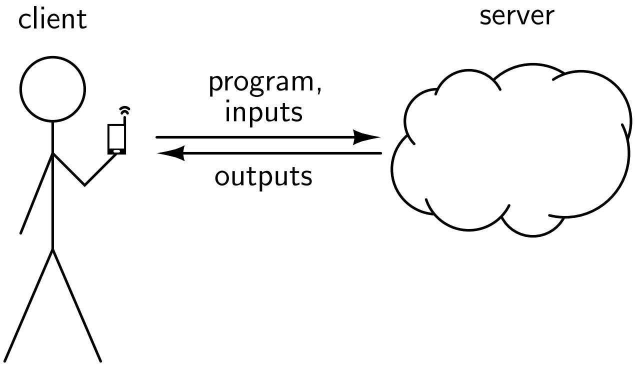
Full accounting for verifiable outsourcing

Riad S. Wahby[⋆], Ye Ji[∘], Andrew J. Blumberg[†], abhi shelat[‡],
Justin Thaler[△], Michael Walfish[∘], and Thomas Wies[∘]

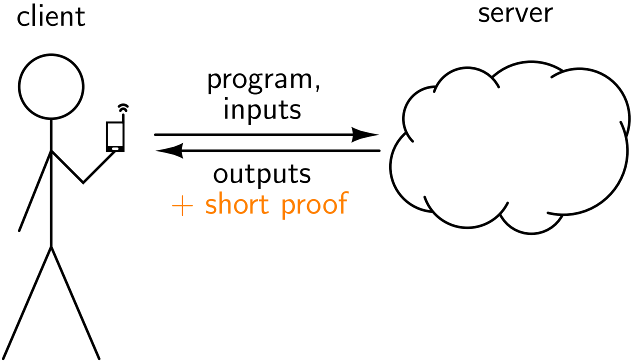
[⋆]Stanford University
[∘]New York University
[†]The University of Texas at Austin
[‡]Northeastern University
[△]Georgetown University

July 6th, 2017

Probabilistic proofs enable outsourcing



Probabilistic proofs enable outsourcing

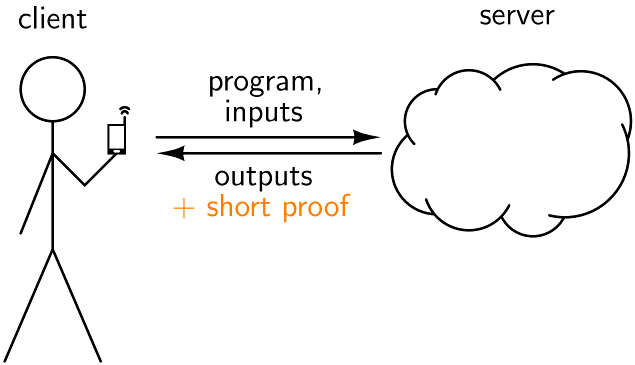


Approach: Server's response includes short proof of correctness.

[Babai85, GMR85, BCC86, BFLS91, FGLSS91, ALMSS92, AS92, Kilian92, LFKN92, Shamir92, Micali00, BG02, BS05, GOS06, BGHSV06, IKO07, GKR08, KR09, GGP10, Groth10, GLR11, Lipmaa11, BCCT12, GGPR13, BCCT13, Thaler13, KRR14, . . .]

Probabilistic proofs enable outsourcing

SBW11
CMT12
SMBW12
TRMP12
SVPBBW12
SBVBPW13
VSBW13
PGHR13
BCGTV13
BFRSBW13
BFR13
DFKP13
BCTV14a
BCTV14b



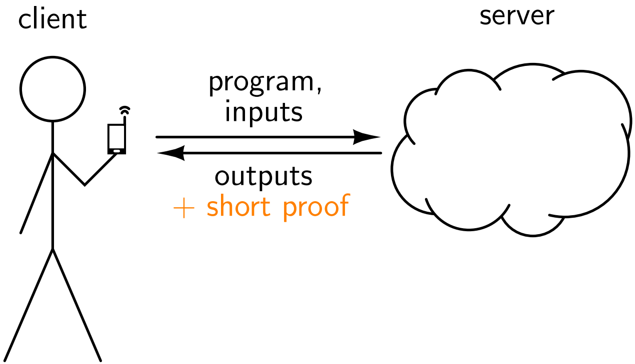
BCGGMTV14
FL14
KPPSST14
FTP14
WSRHBW15
BBFR15
CFHKNPZ15
CTV15
KZMQCPPsS15
D-LFKP16
NT16
ZGKPP17
...

Approach: Server's response includes short proof of correctness.

[Babai85, GMR85, BCC86, BFLS91, FGLSS91, ALMSS92, AS92, Kilian92, LFKN92, Shamir92, Micali00, BG02, BS05, GOS06, BGHSV06, IKO07, GKR08, KR09, GGP10, Groth10, GLR11, Lipmaa11, BCCT12, GGPR13, BCCT13, Thaler13, KRR14, ...]

Probabilistic proofs enable outsourcing

SBW11
CMT12
SMBW12
TRMP12
SVPBBW12
SBVBPW13
VSBW13
PGHR13
BCGTV13
BFRSBW13
BFR13
DFKP13
BCTV14a
BCTV14b



BCGGMTV14
FL14
KPPSST14
FTP14
WSRHBW15
BBFR15
CFHKNPZ15
CTV15
KZMQCPPsS15
D-LFKP16
NT16
ZGKPP17
...

Goal: outsourcing should be less expensive than just executing the computation

Do systems achieve this goal?

Verifier: can easily check proof (asymptotically)

Do systems achieve this goal?

Verifier: can easily check proof (asymptotically)

Prover: has massive overhead ($\approx$10,000,000$\times$)

Do systems achieve this goal?

Verifier: can easily check proof (asymptotically)

Prover: has massive overhead ($\approx$10,000,000$\times$)

Precomputation: proportional to computation size

Do systems achieve this goal?

Verifier: can easily check proof (asymptotically)

Prover: has massive overhead ($\approx$10,000,000$\times$)

Precomputation: proportional to computation size

How do systems handle these costs?

Do systems achieve this goal?

Verifier: can easily check proof (asymptotically)

Prover: has massive overhead ($\approx$10,000,000$\times$)

Precomputation: proportional to computation size

How do systems handle these costs?

Precomputation: amortize over many instances

Do systems achieve this goal?

Verifier: can easily check proof (asymptotically)

Prover: has massive overhead ($\approx$10,000,000$\times$)

Precomputation: proportional to computation size

How do systems handle these costs?

Precomputation: amortize over many instances

Prover: assume $> 10^8\times$ cheaper than verifier

Our contribution

Giraffe: first system to consider all costs and win.

Our contribution

Giraffe: first system to consider all costs and win.

In Giraffe, $\mathcal{P}$ really is $10^8\times$ cheaper than $\mathcal{V}$!
(setting: building trustworthy hardware)

Our contribution

Giraffe: first system to consider all costs and win.

In Giraffe, $\mathcal{P}$ really is $10^8\times$ cheaper than $\mathcal{V}$!
(setting: building trustworthy hardware)

Giraffe extends Zebra [WHGsW, Oakland16] with:

- an asymptotically optimal proof protocol that improves on prior work [Thaler, CRYPTO13]
- a compiler that generates optimized hardware designs from a subset of C

Our contribution

Giraffe: first system to consider all costs and win.

In Giraffe, $\mathcal{P}$ really is $10^8\times$ cheaper than $\mathcal{V}$!
(setting: building trustworthy hardware)

Giraffe extends Zebra [WHGsW, Oakland16] with:

- an asymptotically optimal proof protocol that improves on prior work [Thaler, CRYPTO13]
- a compiler that generates optimized hardware designs from a subset of C

Bottom line: Giraffe makes outsourcing worthwhile

Our contribution

Giraffe: first system to consider all costs and win.

In Giraffe, $\mathcal{P}$ really is $10^8\times$ cheaper than $\mathcal{V}$!
(setting: building trustworthy hardware)

Giraffe extends Zebra [WHGsW, Oakland16] with:

- an asymptotically optimal proof protocol that improves on prior work [Thaler, CRYPTO13]
- a compiler that generates optimized hardware designs from a subset of C

Bottom line: Giraffe makes outsourcing worthwhile (...sometimes).

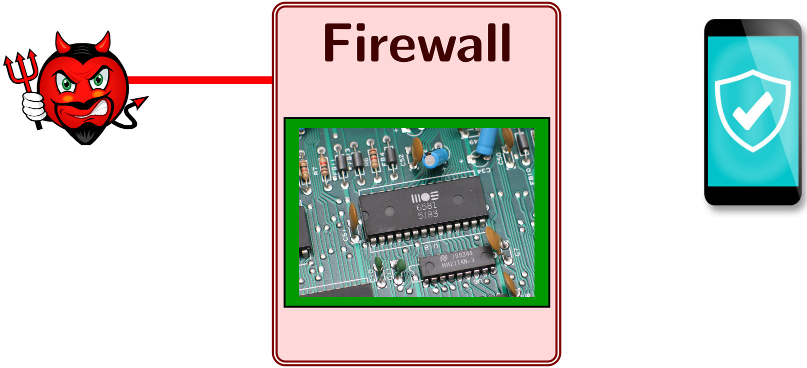
Roadmap

1. Verifiable ASICs
2. Giraffe: a high-level view
3. Evaluation

Roadmap

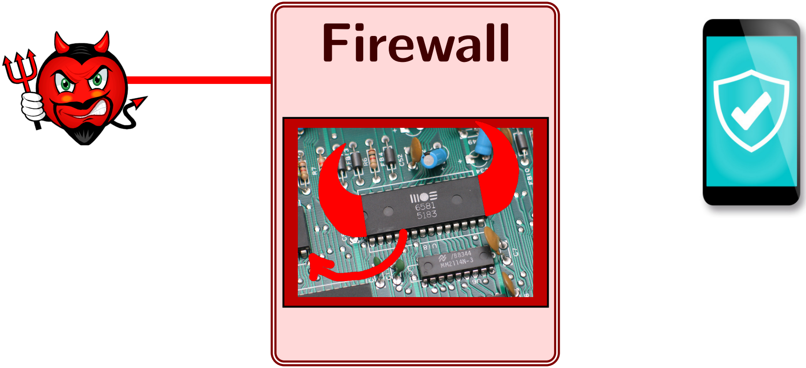
1. Verifiable ASICs
2. Giraffe: a high-level view
3. Evaluation

How can we build trustworthy hardware?



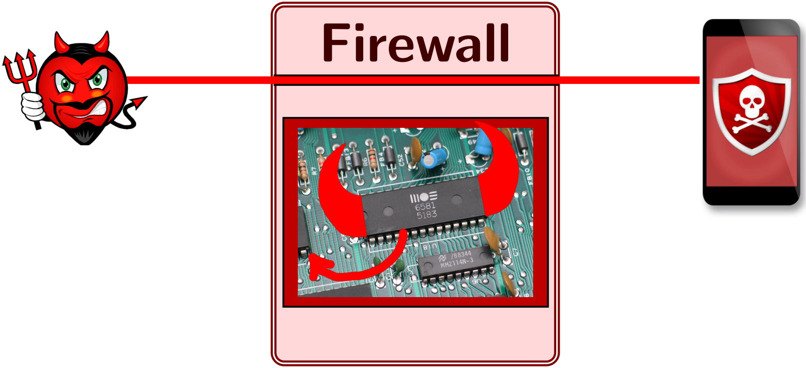
e.g., a custom chip for network packet processing
whose manufacture we outsource to a third party

Untrusted manufacturers can craft hardware Trojans



What if the chip's manufacturer inserts a **back door**?

Untrusted manufacturers can craft hardware Trojans

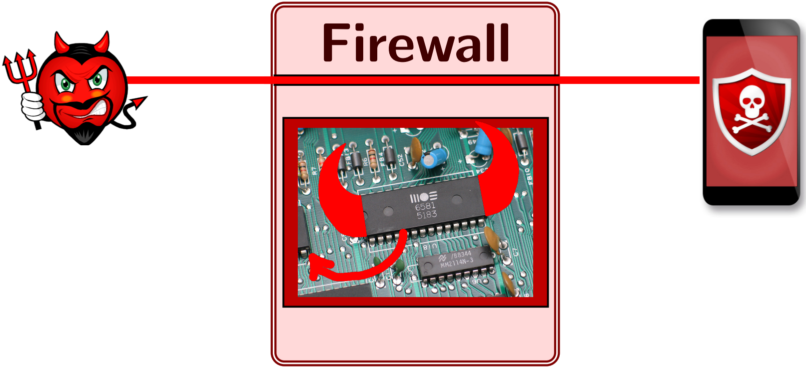


What if the chip's manufacturer inserts a **back door**?

Threat: incorrect execution of the packet filter

(Other concerns, e.g., secret state, are important but orthogonal)

Untrusted manufacturers can craft hardware Trojans



What if the chip's manufacturer inserts a **back door**?

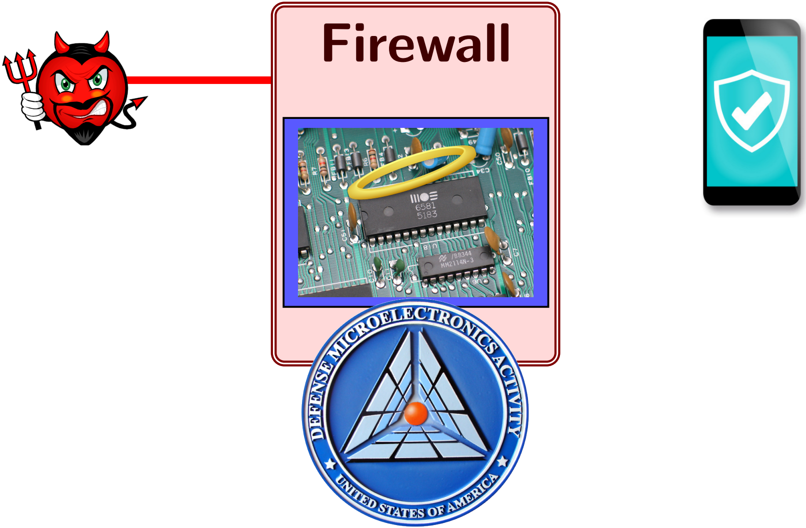
The Cybercrime Economy

Fake tech gear has infiltrated the U.S. government

by David Goldman @DavidGoldmanCNN

November 8, 2012: 3:10 PM ET

Untrusted manufacturers can craft hardware Trojans



US DoD controls supply chain with **trusted foundries**.

Trusted fabs are the only way to get strong guarantees

For example, stealthy trojans can thwart post-fab detection
[A2: Analog Malicious Hardware, Yang et al., Oakland16;
Stealthy Dopant-Level Trojans, Becker et al., CHES13]

Trusted fabs are the only way to get strong guarantees

For example, stealthy trojans can thwart post-fab detection
[A2: Analog Malicious Hardware, Yang et al., Oakland16; Stealthy Dopant-Level Trojans, Becker et al., CHES13]

But trusted fabrication is not a panacea:

- ✗ Only 5 countries have cutting-edge fabs on-shore
- ✗ Building a new fab takes $$$$$$, years of R&D

Trusted fabs are the only way to get strong guarantees

For example, stealthy trojans can thwart post-fab detection
[A2: Analog Malicious Hardware, Yang et al., Oakland16;
Stealthy Dopant-Level Trojans, Becker et al., CHES13]

But trusted fabrication is not a panacea:

- ✗ Only 5 countries have cutting-edge fabs on-shore
- ✗ Building a new fab takes $$$$$$, years of R&D
- ✗ Semiconductor scaling: chip area and energy go with square and cube of transistor length ("critical dimension")
- ✗ So using an old fab means an enormous performance hit
  e.g., India's best on-shore fab is $10^8\times$ behind state of the art

Trusted fabs are the only way to get strong guarantees

For example, stealthy trojans can thwart post-fab detection
[A2: Analog Malicious Hardware, Yang et al., Oakland16; Stealthy Dopant-Level Trojans, Becker et al., CHES13]

But trusted fabrication is not a panacea:

- ✗ Only 5 countries have cutting-edge fabs on-shore
- ✗ Building a new fab takes $$$$$$, years of R&D
- ✗ Semiconductor scaling: chip area and energy go with square and cube of transistor length ("critical dimension")
- ✗ So using an old fab means an enormous performance hit
  e.g., India's best on-shore fab is $10^8\times$ behind state of the art

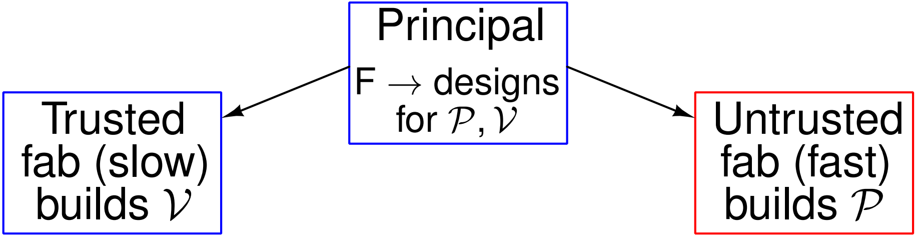
Idea: outsource computations to untrusted chips

Verifiable ASICs [WHGsW16]

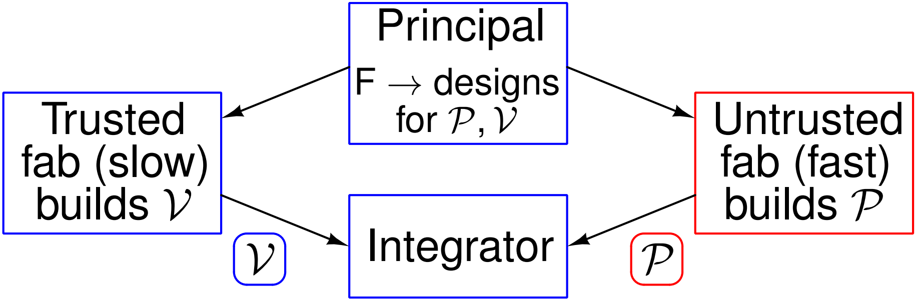
Principal

F $\rightarrow$ designs for $\mathcal{P}, \mathcal{V}$

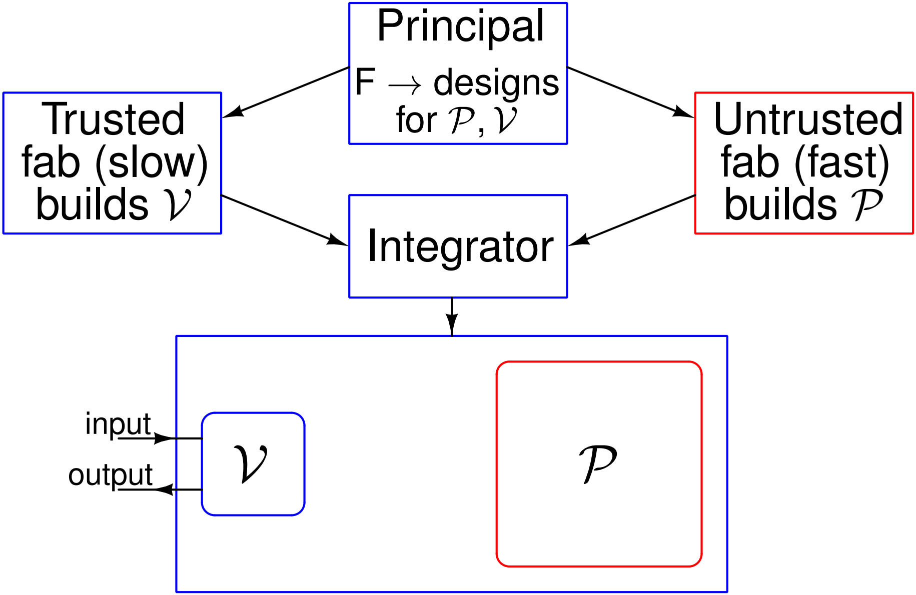
Verifiable ASICs [WHGsW16]



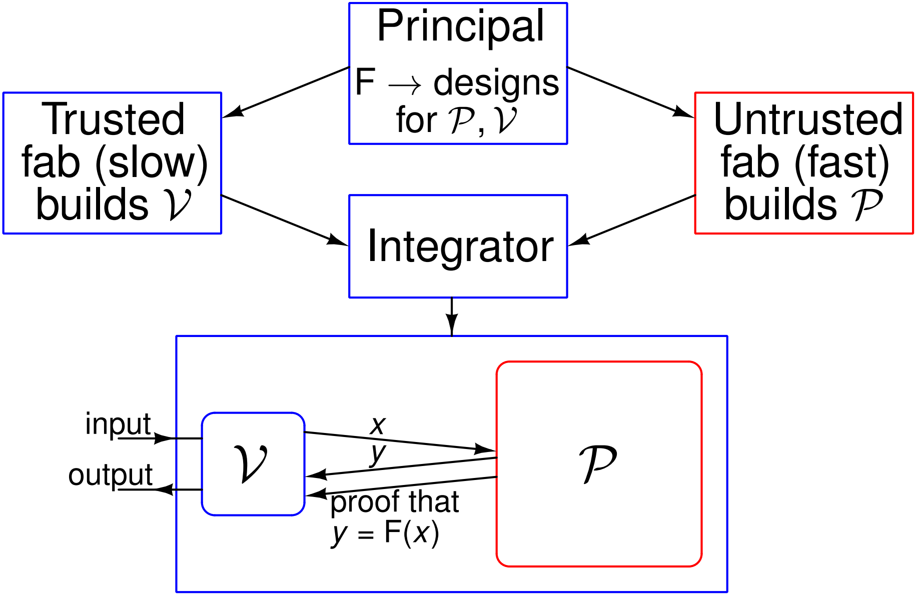
Verifiable ASICs [WHGsW16]



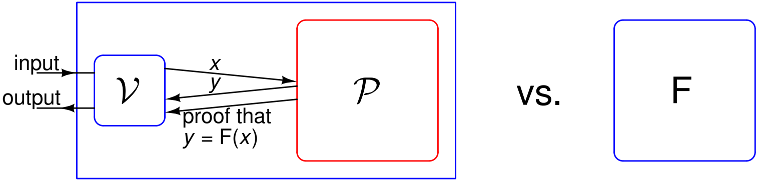
Verifiable ASICs [WHGsW16]



Verifiable ASICs [WHGsW16]

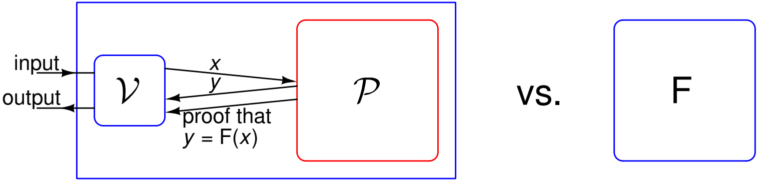


Can Verifiable ASICs be practical?



$\mathcal{V}$ overhead: checking proof is cheap

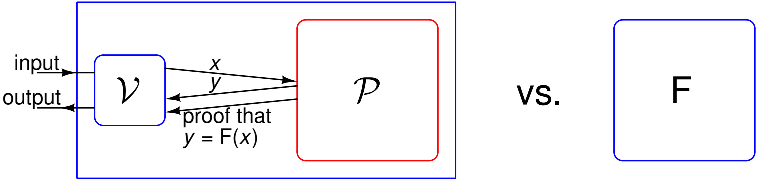
Can Verifiable ASICs be practical?



$\mathcal{V}$ overhead: checking proof is cheap

$\mathcal{P}$ overhead: high compared to cost of F...

Can Verifiable ASICs be practical?

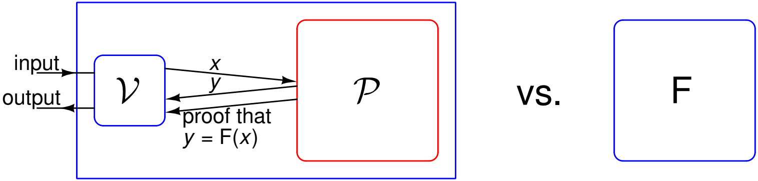


$\mathcal{V}$ overhead: checking proof is cheap

$\mathcal{P}$ overhead: high compared to cost of F...

...but $\mathcal{P}$ uses an advanced circuit technology

Can Verifiable ASICs be practical?



$\mathcal{V}$ overhead: checking proof is cheap

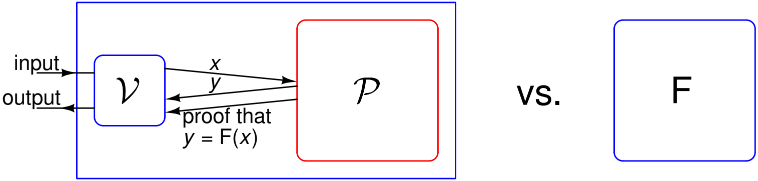
$\mathcal{P}$ overhead: high compared to cost of F...

...but $\mathcal{P}$ uses an advanced circuit technology

Prior work:

$\mathcal{V} + \mathcal{P} < \mathsf{F}$

Can Verifiable ASICs be practical?



$\mathcal{V}$ overhead: checking proof is cheap

$\mathcal{P}$ overhead: high compared to cost of F...

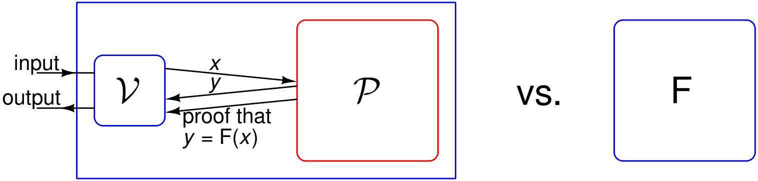
...but $\mathcal{P}$ uses an advanced circuit technology

Precomputation: proportional to cost of F

Prior work:

$\mathcal{V} + \mathcal{P} + \text{Precomp} > \text{F}$

Can Verifiable ASICs be practical?



$\mathcal{V}$ overhead: checking proof is cheap

$\mathcal{P}$ overhead: high compared to cost of F...

...but $\mathcal{P}$ uses an advanced circuit technology

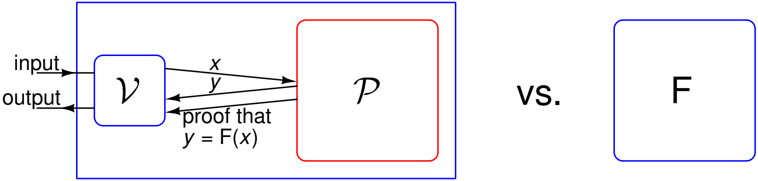
Precomputation: proportional to cost of F

Prior work assumes this away

Prior work:

$\mathcal{V} + \mathcal{P} + \text{Precomp} > \text{F}$

Can Verifiable ASICs be practical?



$\mathcal{V}$ overhead: checking proof is cheap

$\mathcal{P}$ overhead: high compared to cost of F...

...but $\mathcal{P}$ uses an advanced circuit technology

Precomputation: proportional to cost of F

Prior work assumes this away

Our goal:

$\mathcal{V} + \mathcal{P} + \text{Precomp} < \text{F}$

Roadmap

1. Verifiable ASICs
2. Giraffe: a high-level view
3. Evaluation

Evolution of Giraffe's back-end

GKR08 base protocol

Evolution of Giraffe's back-end

GKR08 base protocol

CMT12 reduces $\mathcal{P}$ and precomp costs for all ckts

Evolution of Giraffe's back-end

GKR08 base protocol

CMT12 reduces $\mathcal{P}$ and precomp costs for all ckts

Thaler13 reduces precomp for structured circuits

Evolution of Giraffe's back-end

GKR08 base protocol

CMT12 reduces $\mathcal{P}$ and precomp costs for all ckts

Thaler13 reduces precomp for structured circuits

Giraffe reduces $\mathcal{P}$ cost for structured circuits
(plus optimizations for $\mathcal{V}$; see paper)

Evolution of Giraffe's back-end

GKR08 base protocol

CMT12 reduces $\mathcal{P}$ and precomp costs for all ckts

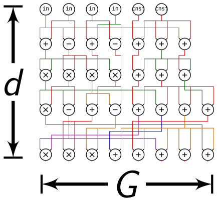
Thaler13 reduces precomp for structured circuits

Giraffe reduces $\mathcal{P}$ cost for structured circuits
(plus optimizations for $\mathcal{V}$; see paper)

Let's take a high-level look at how these optimizations work.
(The following all use a nice simplification [Thaler15].)

GKR08 (a quick reminder)

For each layer of an arithmetic circuit, $\mathcal{P}$ and $\mathcal{V}$ engage in a sum-check protocol.

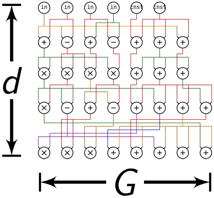


GKR08 (a quick reminder)

For each layer of an arithmetic circuit, $\mathcal{P}$ and $\mathcal{V}$ engage in a sum-check protocol.

In the first round, $\mathcal{P}$ computes ($q \in \mathbb{F}^{\log G}$):

$$\sum_{h_0 \in \{0,1\}^{\log G}} \sum_{h_1 \in \{0,1\}^{\log G}} \left(\tilde{\text{add}}(q, h_0, h_1) \left(\tilde{\text{V}}(h_0) + \tilde{\text{V}}(h_1) \right) + \tilde{\text{mul}}(q, h_0, h_1) \left(\tilde{\text{V}}(h_0) \cdot \tilde{\text{V}}(h_1) \right) \right)$$



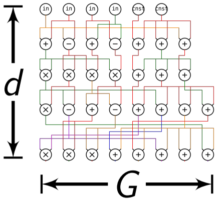
GKR08 (a quick reminder)

For each layer of an arithmetic circuit, $\mathcal{P}$ and $\mathcal{V}$ engage in a sum-check protocol.

In the first round, $\mathcal{P}$ computes ($q \in \mathbb{F}^{\log G}$):

$$\sum_{h_0 \in \{0,1\}^{\log G}} \sum_{h_1 \in \{0,1\}^{\log G}} \left(\tilde{\text{add}}(q, h_0, h_1) \left(\tilde{\text{V}}(h_0) + \tilde{\text{V}}(h_1) \right) + \tilde{\text{mul}}(q, h_0, h_1) \left(\tilde{\text{V}}(h_0) \cdot \tilde{\text{V}}(h_1) \right) \right)$$

This has $2^{2 \log G} = G^2$ terms. In total, $\mathcal{P}$'s work is $\text{O}(\text{poly}(G))$.



GKR08 (a quick reminder)

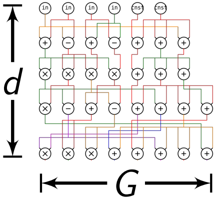
For each layer of an arithmetic circuit, $\mathcal{P}$ and $\mathcal{V}$ engage in a sum-check protocol.

In the first round, $\mathcal{P}$ computes ($q \in \mathbb{F}^{\log G}$):

$$\sum_{h_0 \in \{0,1\}^{\log G}} \sum_{h_1 \in \{0,1\}^{\log G}} \left(\tilde{\text{add}}(q, h_0, h_1) \left(\tilde{\text{V}}(h_0) + \tilde{\text{V}}(h_1) \right) + \tilde{\text{mul}}(q, h_0, h_1) \left(\tilde{\text{V}}(h_0) \cdot \tilde{\text{V}}(h_1) \right) \right)$$

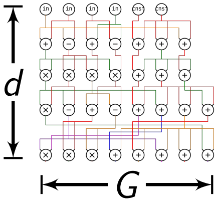
This has $2^{2 \log G} = G^2$ terms. In total, $\mathcal{P}$'s work is O(poly(G)).

Precomputation is one evaluation of $\tilde{\text{add}}$ and $\tilde{\text{mul}}$, costing O(poly(G)).



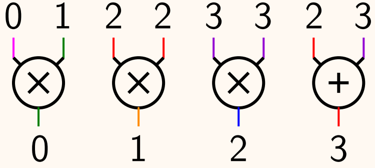
CMT12: from polynomial to quasilinear

$\text{add}(g_O, g_L, g_R) = 0$ except when g_O is $+$ with inputs g_L, g_R

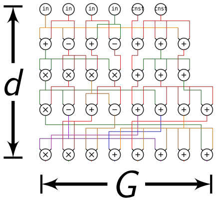


CMT12: from polynomial to quasilinear

$\mathrm{add}(g_O, g_L, g_R) = 0$ except when g_O is $+$ with inputs g_L, g_R



$\mathrm{add}(3, 2, 3) = 1$, otherwise $\mathrm{add}(\cdots) = 0$

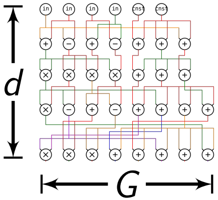


CMT12: from polynomial to quasilinear

$\text{add}(g_O, g_L, g_R) = 0$ except when g_O is $+$ with inputs g_L, g_R

This means we can rewrite $\mathcal{P}$'s sum in the first round as:

$$\sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1)\left(\tilde{\text{V}}(h_0) + \tilde{\text{V}}(h_1)\right) +$$
$$\sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1)\left(\tilde{\text{V}}(h_0) \cdot \tilde{\text{V}}(h_1)\right)$$



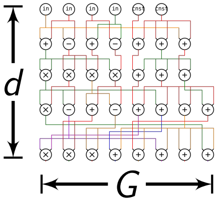
CMT12: from polynomial to quasilinear

$\text{add}(g_O, g_L, g_R) = 0$ except when g_O is $+$ with inputs g_L, g_R

This means we can rewrite $\mathcal{P}$'s sum in the first round as:

$$\sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1)\left(\tilde{\text{V}}(h_0) + \tilde{\text{V}}(h_1)\right) + \sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1)\left(\tilde{\text{V}}(h_0) \cdot \tilde{\text{V}}(h_1)\right)$$

G terms/round for $2\log G$ rounds: $\mathcal{P}$'s work is $\text{O}(G \log G)$.



CMT12: from polynomial to quasilinear

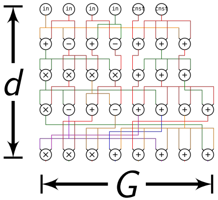
$\text{add}(g_O, g_L, g_R) = 0$ except when g_O is $+$ with inputs g_L, g_R

This means we can rewrite $\mathcal{P}$'s sum in the first round as:

$$\sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1)\left(\tilde{\text{V}}(h_0) + \tilde{\text{V}}(h_1)\right) + \sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1)\left(\tilde{\text{V}}(h_0) \cdot \tilde{\text{V}}(h_1)\right)$$

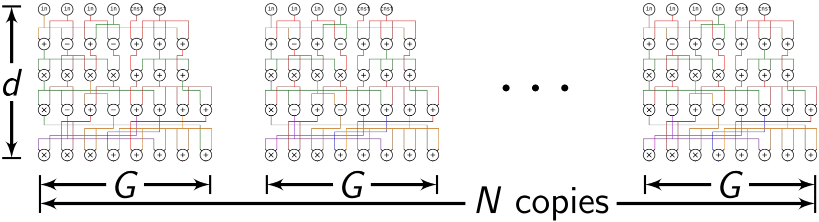
G terms/round for $2 \log G$ rounds: $\mathcal{P}$'s work is $\text{O}(G \log G)$.

Using a related trick, precomputing $\tilde{\text{add}}$ and $\tilde{\text{mul}}$ costs $\text{O}(G)$ in total.



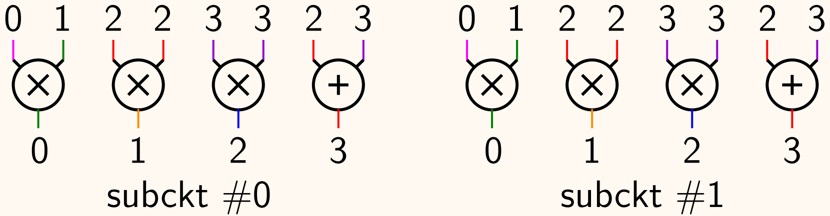
Thaler13: more structure, less precomputation

Idea: for a *batch* of identical subckts, $\tilde{\text{add}}$ and $\tilde{\text{mul}}$ can be "small."



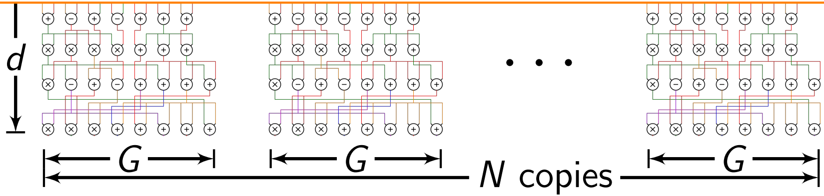
Thaler13: more structure, less precomputation

Idea: for a *batch* of identical subckts, $\tilde{\text{add}}$ and $\tilde{\text{mul}}$ can be "small."



$\text{add}(3,2,3) = 1$, otherwise $\text{add}(\cdots) = 0$

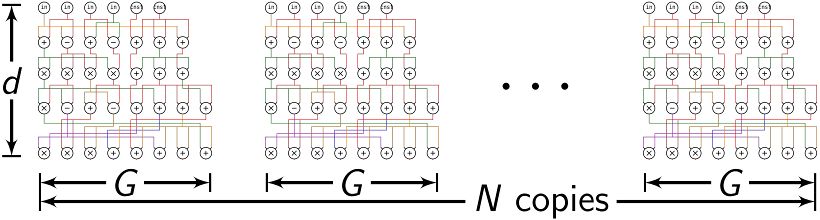
Notice that $\tilde{\text{add}}$ does not comprehend subcircuit number!



Thaler13: more structure, less precomputation

Idea: for a *batch* of identical subckts, $\tilde{\text{add}}$ and $\tilde{\text{mul}}$ can be "small."

➔ Precomp costs $O(G)$, amortized over N copies!



Thaler13: more structure, less precomputation

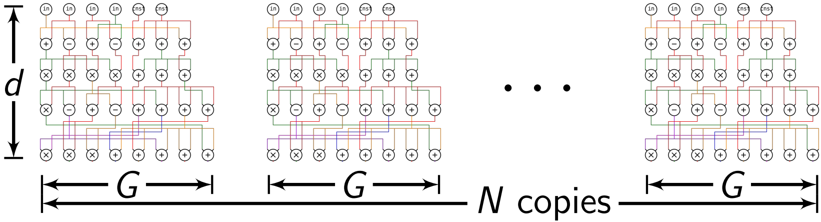
Idea: for a *batch* of identical subckts, $\tilde{\text{add}}$ and $\tilde{\text{mul}}$ can be "small."

➔ Precomp costs O(G), amortized over N copies!

Now $\mathcal{P}$'s sum in the first round is ($q' \in \mathbb{F}^{\log N}$):

$$\sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1) \sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right)\left(\tilde{\text{V}}(h', h_0) + \tilde{\text{V}}(h', h_1)\right) +$$

$$\sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1) \sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right)\left(\tilde{\text{V}}(h', h_0) \cdot \tilde{\text{V}}(h', h_1)\right)$$



Thaler13: more structure, less precomputation

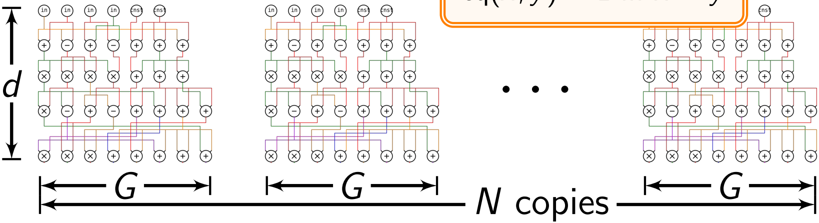
Idea: for a *batch* of identical subckts, $\tilde{\text{add}}$ and $\tilde{\text{mul}}$ can be "small."
➔ Precomp costs $O(G)$, amortized over N copies!

Now $\mathcal{P}$'s sum in the first round is ($q' \in \mathbb{F}^{\log N}$):

$$\sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1) \sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \left(\tilde{V}(h', h_0) + \tilde{V}(h', h_1)\right) +$$

$$\sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1) \sum_{h' \in \{0,1\}^{\log N}} \boxed{\tilde{\text{eq}}\left(q', h'\right)} \left(\tilde{V}(h', h_0) \cdot \tilde{V}(h', h_1)\right)$$

eq$(x, y) = 1$ iff $x = y$



Thaler13: more structure, less precomputation

Idea: for a *batch* of identical subckts, $\tilde{\text{add}}$ and $\tilde{\text{mul}}$ can be "small."

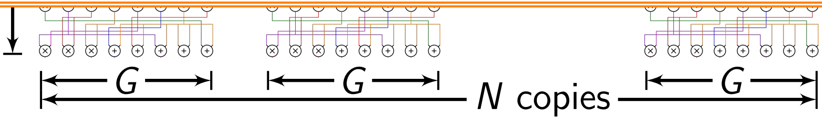
➔ Precomp costs $O(G)$, amortized over N copies!

Now $\mathcal{P}$'s sum in the first round is ($q' \in \mathbb{F}^{\log N}$):

$$\sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1) \sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right)\left(\tilde{V}(h', h_0) + \tilde{V}(h', h_1)\right) +$$

$$\sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1) \sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right)\left(\tilde{V}(h', h_0) \cdot \tilde{V}(h', h_1)\right)$$

For each gate,



Thaler13: more structure, less precomputation

Idea: for a *batch* of identical subckts, $\tilde{\text{add}}$ and $\tilde{\text{mul}}$ can be "small."

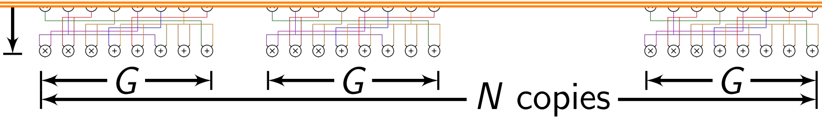
➔ Precomp costs $O(G)$, amortized over N copies!

Now $\mathcal{P}$'s sum in the first round is $(q' \in \mathbb{F}^{\log N})$:

$$\sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1) \sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right)\left(\tilde{V}(h', h_0) + \tilde{V}(h', h_1)\right) +$$

$$\sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1) \sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right)\left(\tilde{V}(h', h_0) \cdot \tilde{V}(h', h_1)\right)$$

For each gate, sum over each subcircuit.



Thaler13: more structure, less precomputation

Idea: for a *batch* of identical subckts, $\tilde{\text{add}}$ and $\tilde{\text{mul}}$ can be "small."

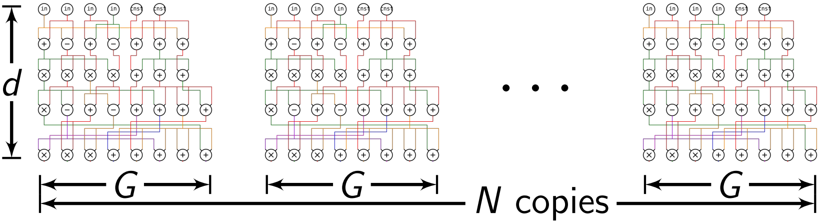
➔ Precomp costs $O(G)$, amortized over N copies!

Now $\mathcal{P}$'s sum in the first round is ($q' \in \mathbb{F}^{\log N}$):

$$\sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1) \sum_{h'\in\{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \left(\tilde{\text{V}}(h', h_0) + \tilde{\text{V}}(h', h_1)\right) +$$

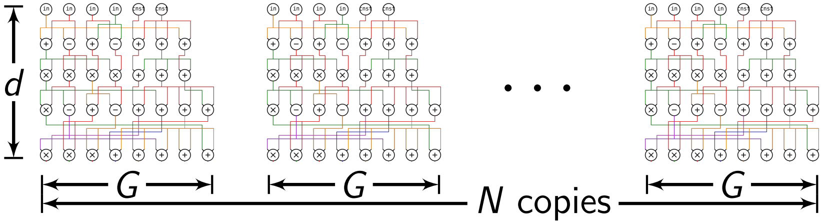
$$\sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1) \sum_{h'\in\{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \left(\tilde{\text{V}}(h', h_0) \cdot \tilde{\text{V}}(h', h_1)\right)$$

NG terms/round in first $2 \log G$ rounds: $\mathcal{P}$'s work is $\Omega(NG \log G)$.



Giraffe: leveraging structure to reduce $\mathcal{P}$ costs

Idea: arrange for copies to "collapse" during sum-check protocol.



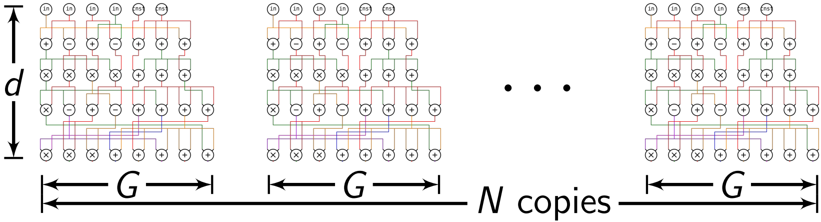
Giraffe: leveraging structure to reduce $\mathcal{P}$ costs

Idea: arrange for copies to "collapse" during sum-check protocol.

Rewriting the prior sum and changing sumcheck order:

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1) \in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1) \left(\tilde{\text{V}}(h', h_0) + \tilde{\text{V}}(h', h_1)\right) +$$

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1) \in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1) \left(\tilde{\text{V}}(h', h_0) \cdot \tilde{\text{V}}(h', h_1)\right)$$



Giraffe: leveraging structure to reduce $\mathcal{P}$ costs

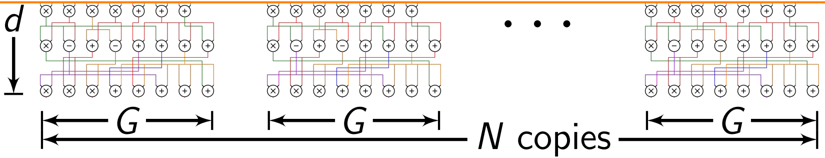
Idea: arrange for copies to "collapse" during sum-check protocol.

Rewriting the prior sum and changing sumcheck order:

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0, h_1) \in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1)\left(\tilde{\text{V}}(h', h_0) + \tilde{\text{V}}(h', h_1)\right) +$$

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0, h_1) \in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1)\left(\tilde{\text{V}}(h', h_0) \cdot \tilde{\text{V}}(h', h_1)\right)$$

For each subcircuit,



Giraffe: leveraging structure to reduce $\mathcal{P}$ costs

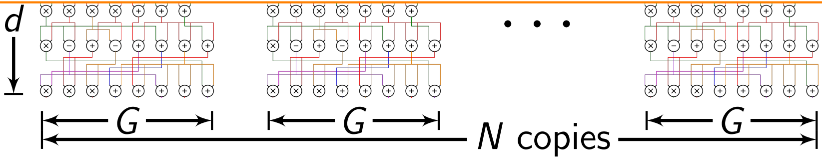
Idea: arrange for copies to "collapse" during sum-check protocol.

Rewriting the prior sum and changing sumcheck order:

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1)\left(\tilde{\text{V}}(h', h_0) + \tilde{\text{V}}(h', h_1)\right) +$$

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1)\left(\tilde{\text{V}}(h', h_0) \cdot \tilde{\text{V}}(h', h_1)\right)$$

For each subcircuit, sum over each gate.



Giraffe: leveraging structure to reduce $\mathcal{P}$ costs

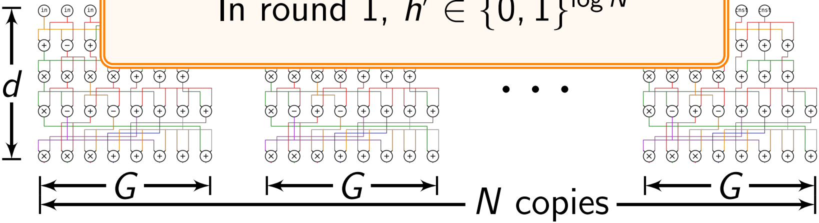
Idea: arrange for copies to "collapse" during sum-check protocol.

Rewriting the prior sum and changing sumcheck order:

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0, h_1) \in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1) \left(\tilde{V}(h', h_0) + \tilde{V}(h', h_1)\right) +$$

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0, h_1) \in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1) \left(\tilde{V}(h', h_0) \cdot \tilde{V}(h', h_1)\right)$$

In round 1, $h' \in \{0, 1\}^{\log N}$



Giraffe: leveraging structure to reduce $\mathcal{P}$ costs

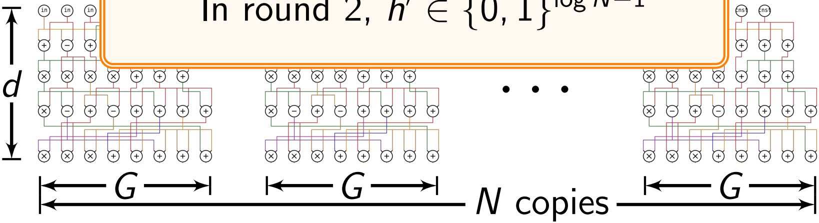
Idea: arrange for copies to "collapse" during sum-check protocol.

Rewriting the prior sum and changing sumcheck order:

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1) \in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1) \left(\tilde{\text{V}}(h', h_0) + \tilde{\text{V}}(h', h_1)\right) +$$

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1) \in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1) \left(\tilde{\text{V}}(h', h_0) \cdot \tilde{\text{V}}(h', h_1)\right)$$

In round 2, $h' \in \{0,1\}^{\log N - 1}$



Giraffe: leveraging structure to reduce $\mathcal{P}$ costs

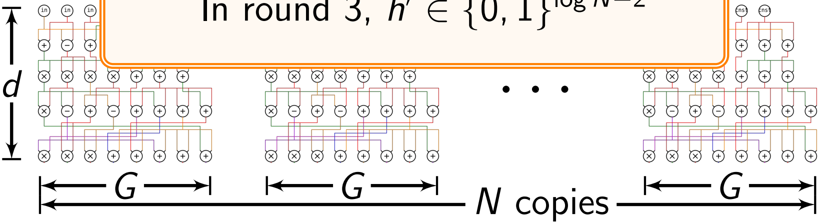
Idea: arrange for copies to "collapse" during sum-check protocol.

Rewriting the prior sum and changing sumcheck order:

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1) \in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1)\left(\tilde{\text{V}}(h', h_0) + \tilde{\text{V}}(h', h_1)\right) +$$

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1) \in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1)\left(\tilde{\text{V}}(h', h_0) \cdot \tilde{\text{V}}(h', h_1)\right)$$

In round 3, $h' \in \{0,1\}^{\log N - 2}$



Giraffe: leveraging structure to reduce $\mathcal{P}$ costs

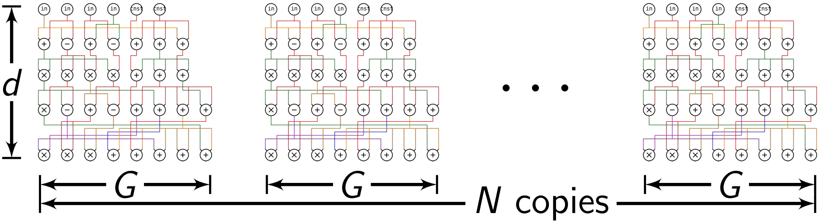
Idea: arrange for copies to "collapse" during sum-check protocol.

Rewriting the prior sum and changing sumcheck order:

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1)\in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1)\left(\tilde{\text{V}}(h', h_0) + \tilde{\text{V}}(h', h_1)\right) +$$

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1)\in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1)\left(\tilde{\text{V}}(h', h_0) \cdot \tilde{\text{V}}(h', h_1)\right)$$

$\mathcal{P}$ does $\left(N + \frac{N}{2} + \frac{N}{4} + \ldots\right) G + 2G \log G = \text{O}(NG + G \log G)$ work.



Giraffe: leveraging structure to reduce $\mathcal{P}$ costs

Idea: arrange for copies to "collapse" during sum-check protocol.

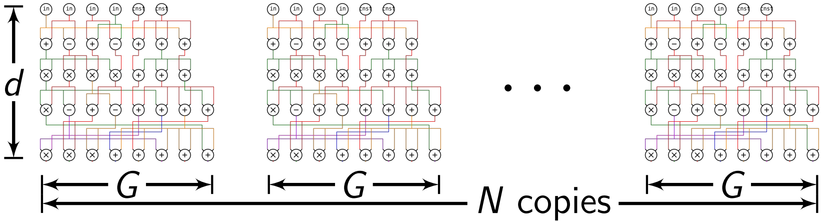
Rewriting the prior sum and changing sumcheck order:

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1) \in S_{\text{add}}} \tilde{\text{add}}(q, h_0, h_1) \left(\tilde{V}(h', h_0) + \tilde{V}(h', h_1)\right) +$$

$$\sum_{h' \in \{0,1\}^{\log N}} \tilde{\text{eq}}\left(q', h'\right) \sum_{(h_0,h_1) \in S_{\text{mul}}} \tilde{\text{mul}}(q, h_0, h_1) \left(\tilde{V}(h', h_0) \cdot \tilde{V}(h', h_1)\right)$$

$\mathcal{P}$ does $\left(N + \frac{N}{2} + \frac{N}{4} + \ldots\right) G + 2G \log G = \mathrm{O}(NG + G \log G)$ work.

➔ Linear in size of computation when $N > \log G$!



Roadmap

1. Verifiable ASICs
2. Giraffe: a high-level view
3. Evaluation

Implementation

Giraffe is an end-to-end hardware generator:

Implementation

Giraffe is an end-to-end hardware generator:

a hardware *design template*

given computation, chip parameters (technology, size, . . .),
produces optimized hardware designs for $\mathcal{P}$ and $\mathcal{V}$

Implementation

Giraffe is an end-to-end hardware generator:

a hardware *design template*

given computation, chip parameters (technology, size, . . .), produces optimized hardware designs for $\mathcal{P}$ and $\mathcal{V}$

a (subset of) C compiler

produces the representation used by the design template

Evaluation questions

How does Giraffe perform on real-world computations?

1. Curve25519 point multiplication
2. Image matching

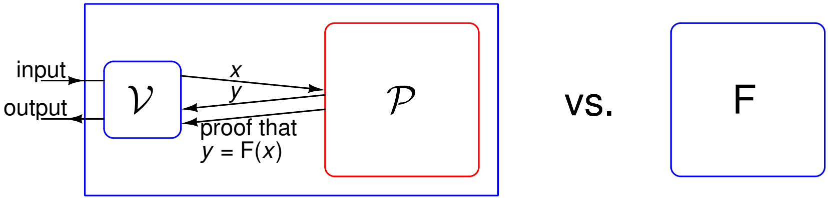
Evaluation questions

How does Giraffe perform on real-world computations?

1. Curve25519 point multiplication
2. Image matching

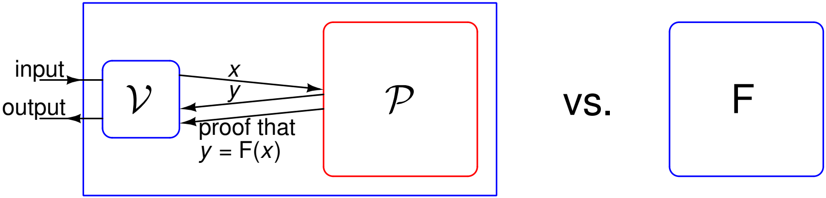
Goal: total cost of $\mathcal{V}$, $\mathcal{P}$, and precomputation should be less than building F on a trusted chip

Evaluation method



Baselines: Zebra; implementation of F in same technology as $\mathcal{V}$

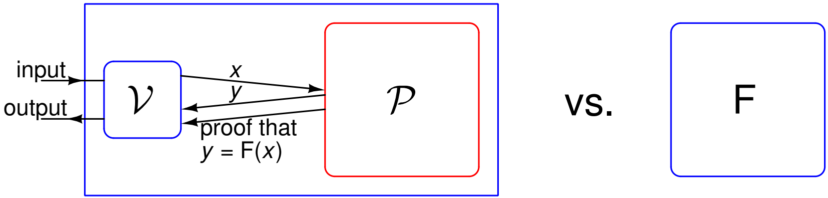
Evaluation method



Baselines: Zebra; implementation of F in same technology as $\mathcal{V}$

Metric: total energy consumption

Evaluation method



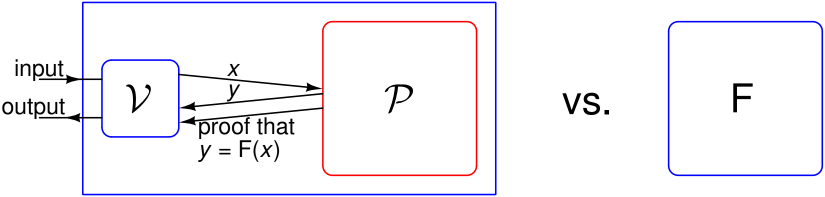
Baselines: Zebra; implementation of F in same technology as $\mathcal{V}$

Metric: total energy consumption

Measurements: based on circuit synthesis and simulation, published chip designs, and CMOS scaling models

Charge for $\mathcal{V}$, $\mathcal{P}$, communication; precomputation; PRNG

Evaluation method



Baselines: Zebra; implementation of F in same technology as $\mathcal{V}$

Metric: total energy consumptio

Measurements: based on circuit
published chip designs, and CM

350 nm: 1997 (Pentium II)
7 nm: $\approx$ 2018
$\approx$ 20 year gap between
trusted and untrusted fab

Charge for $\mathcal{V}$, $\mathcal{P}$, communication; precomputation; PRN

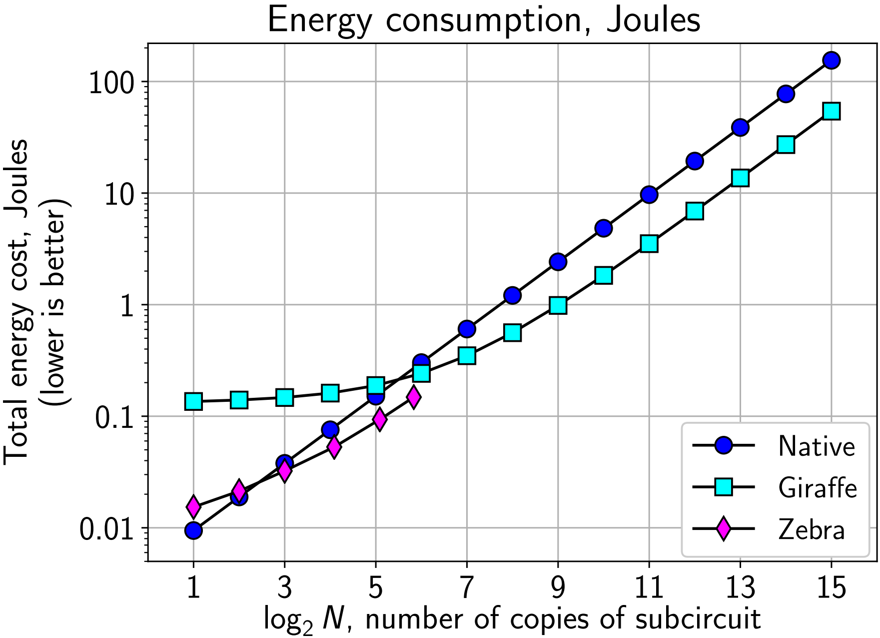
Constraints: trusted fab = 350 nm; untrusted fab = 7 nm
200 mm^2 max chip area; 150 W max total power

Application #1: Curve25519 point multiplication

Curve25519: a commonly-used elliptic curve

Point multiplication: primitive, e.g., for ECDH

Application #1: Curve25519 point multiplication

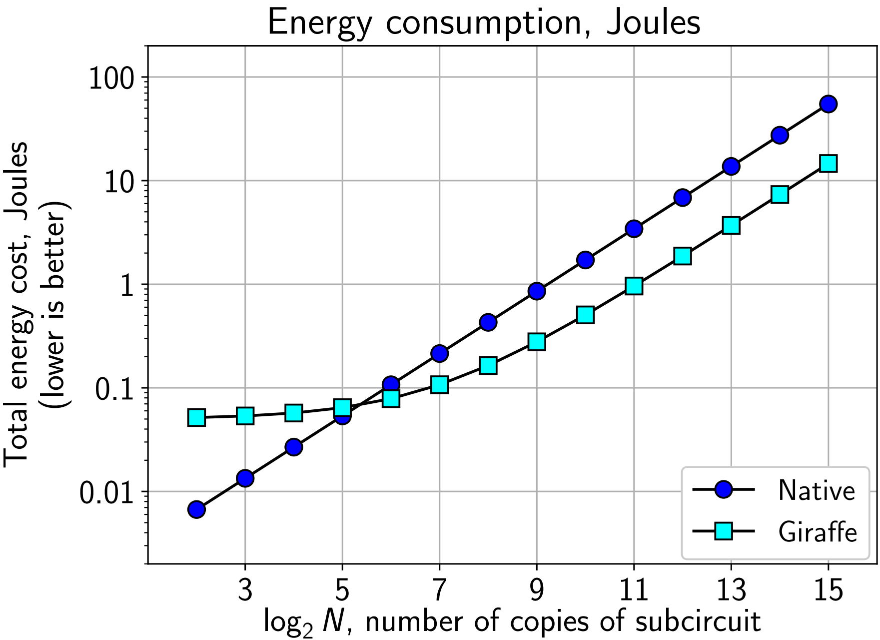


Application #2: Image matching

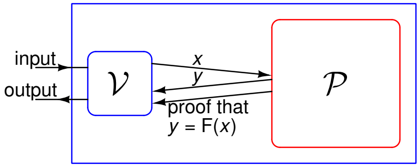
Image matching via Fast Fourier transform

C implementation, compiled by Giraffe's front-end to $\mathcal{V}$ and $\mathcal{P}$ hardware designs—no hand tweaking!

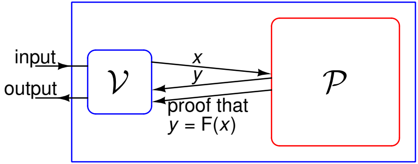
Application #2: Image matching



Recap: is it **practical**?

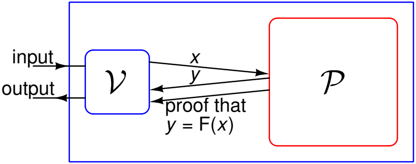


Recap: is it **practical**?



✗ Giraffe is restricted to batched computations

Recap: is it **practical**?



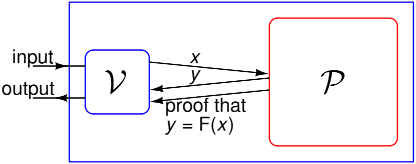
✗ Giraffe is restricted to batched computations

Giraffe's front-end includes two static analysis passes:

Slicing extracts only the parts of programs that can be efficiently outsourced

Squashing extracts batch-parallelism from serial computations

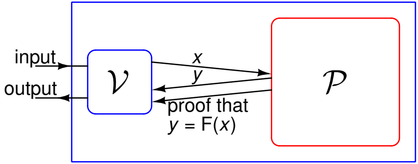
Recap: is it **practical**?



✗ Giraffe is restricted to batched computations

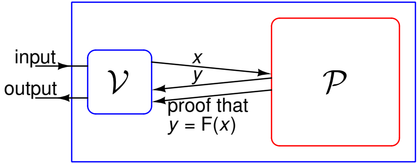
✓ Giraffe's proof protcol and optimizations save orders of magnitude compared to prior work

Recap: is it **practical**?



- ✗ Giraffe is restricted to batched computations
- ✓ Giraffe's proof protcol and optimizations save orders of magnitude compared to prior work
- ✓ Giraffe is the first system in the literature to account for *all costs*—and win.

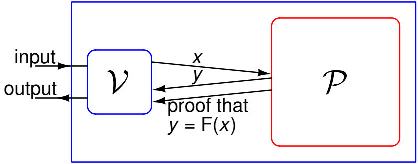
Recap: is it **practical**?



- ✗ Giraffe is restricted to batched computations
- ✓ Giraffe's proof protcol and optimizations save orders of magnitude compared to prior work
- ✓ Giraffe is the first system in the literature to account for *all costs*—and win.

Giraffe is a step, but much work remains!

Recap: is it **practical**?



- ✗ Giraffe is restricted to batched computations
- ✓ Giraffe's proof protcol and optimizations save orders of magnitude compared to prior work
- ✓ Giraffe is the first system in the literature to account for *all costs*—and win.

Giraffe is a step, but much work remains!

https://giraffe.crypto.fyi

http://www.pepper-project.org