Small cryptographic bytecode

D. J. Bernstein

elaborating on an idea from
Adam Langley

“Line search”:
trying to find minimum of function \( f \) defined on \( x \)-line.

e.g. “Bisection”, trying to find minimum in interval \([x_0, x_1]\):
Replace interval with either
\([x_0, (x_0 + x_1)/2]\) or \([(x_0 + x_1)/2, x_1]\);
try to make sensible choice.
Iterate many times.
Small cryptographic bytecode

D. J. Bernstein

elaborating on an idea from
Adam Langley

“Line search”: trying to find minimum of function $f$ defined on $x$-line.

E.g. “Bisection”, trying to find minimum in interval $[x_0, x_1]$: Replace interval with either $[x_0, (x_0+x_1)/2]$ or $[(x_0+x_1)/2, x_1]$; try to make sensible choice.

Iterate many times.

Can try to reduce #iterations using smarter models of $f$: see, e.g., “secant method”.
Small cryptographic bytecode
D. J. Bernstein
elaborating on an idea from
Adam Langley

“Line search”: trying to find minimum of function $f$ defined on $x$-line.

E.g. “Bisection”, trying to find minimum in interval $[x_0, x_1]$:
Replace interval with either $[x_0, (x_0 + x_1)/2]$ or $[(x_0 + x_1)/2, x_1]$;
try to make sensible choice.
Iterate many times.

Can try to reduce #iterations using smarter models of $f$:
see, e.g., “secant method”.

Harder when $f$ varies more.
“Line search”: trying to find minimum of function $f$ defined on $x$-line.

E.g., “Bisection”, trying to find minimum in interval $[x_0, x_1]$:
Replace interval with either $[x_0, (x_0 + x_1)/2]$ or $[(x_0 + x_1)/2, x_1]$; try to make sensible choice.
Iterate many times.

Can try to reduce #iterations using smarter models of $f$: see, e.g., “secant method”.

Harder when $f$ varies more.

How to find minimum of function $f$ defined on $(x; y)$-plane?

“Gradient descent”: Starting from $(x_0; y_0)$, try to figure out direction where $f$ decreases fastest.
Small cryptographic bytecode

D. J. Bernstein elaborating on an idea from Adam Langley

“Line search”:
trying to find minimum of function $f$ defined on $x$-line.
e.g. “Bisection”, trying to find minimum in interval $[x_0, x_1]$:
Replace interval with either $[x_0, (x_0+x_1)/2]$ or $[(x_0+x_1)/2, x_1]$;
try to make sensible choice.
Iterate many times.

Can try to reduce #iterations using smarter models of $f$:
see, e.g., “secant method”.

Harder when $f$ varies more.

How to find minimum of function $f$ defined on $(x, y)$-plane?

“Gradient descent”:
Starting from $(x_0, y_0)$, try to figure out direction
where $f$ decreases fastest.
How to find minimum of function $f$ defined on $(x, y)$-plane?

“Gradient descent”:
Starting from $(x_0, y_0)$, try to figure out direction where $f$ decreases fastest.

“Line search”:
trying to find minimum of function $f$ defined on $x$-line.
e.g. “Bisection”, trying to find minimum in interval $[x_0, x_1]$:
Replace interval with either $[x_0, (x_0 + x_1)/2]$ or $[(x_0 + x_1)/2, x_1]$;
try to make sensible choice.
Iterate many times.

Can try to reduce #iterations using smarter models of $f$:
see, e.g., “secant method”.

Harder when $f$ varies more.
“Line search”:
trying to find minimum of
function $f$ defined on $x$-line.
e.g. “Bisection”, trying to find
minimum in interval $[x_0, x_1]$:
Replace interval with either
$[x_0, (x_0 + x_1)/2]$ or $[(x_0 + x_1)/2, x_1]$;
try to make sensible choice.
Iterate many times.
Can try to reduce #iterations
using smarter models of $f$:
see, e.g., “secant method”.
Harder when $f$ varies more.

How to find minimum of function
$f$ defined on $(x, y)$-plane?

“Gradient descent”:
Starting from $(x_0, y_0)$,
try to figure out direction
where $f$ decreases fastest.
“Line search”: trying to find minimum of function $f$ defined on $x$-line.
e.g. “Bisection”, trying to find minimum in interval $[x_0, x_1]$:
Replace interval with either $[x_0, (x_0+x_1)/2]$ or $[(x_0+x_1)/2, x_1]$;
try to make sensible choice.
Iterate many times.

Can try to reduce #iterations using smarter models of $f$:
see, e.g., “secant method”.

Harder when $f$ varies more.

How to find minimum of function $f$ defined on $(x, y)$-plane?

“Gradient descent”: Starting from $(x_0, y_0)$,
try to figure out direction where $f$ decreases fastest.

Could do line search to find minimum in that direction.
Then find a new direction.
“Line search”:
trying to find minimum of
function $f$ defined on $x$-line.
e.g. “Bisection”, trying to find
minimum in interval $[x_0, x_1]$:
Replace interval with either
$[x_0, (x_0+x_1)/2]$ or $[(x_0+x_1)/2, x_1]$;
try to make sensible choice.
Iterate many times.
Can try to reduce #iterations
using smarter models of $f$:
see, e.g., “secant method”.
Harder when $f$ varies more.

How to find minimum of function
$f$ defined on $(x, y)$-plane?

“Gradient descent”:
Starting from $(x_0, y_0)$,
try to figure out direction
where $f$ decreases fastest.
Could do line search to find
minimum in that direction.
Then find a new direction.
Better: Step down that direction.
Then find a new direction.
“Line search”: trying to find minimum of function $f$ defined on $x$-line.

e.g. “Bisection”, trying to find minimum in interval $[x_0, x_1]$:
Replace interval with either $[x_0, (x_0 + x_1)/2]$ or $[(x_0 + x_1)/2, x_1]$;
try to make sensible choice.
Iterate many times.

Can try to reduce #iterations using smarter models of $f$:
see, e.g., “secant method”.

Harder when $f$ varies more.

How to find minimum of function $f$ defined on $(x, y)$-plane?

“Gradient descent”:
Starting from $(x_0, y_0)$, try to figure out direction where $f$ decreases fastest.
Could do line search to find minimum in that direction.
Then find a new direction.
Better: Step down that direction.
Then find a new direction.
Silly: Line search in $x$ direction; line search in $y$ direction; repeat.
"Line search": trying to find minimum of function $f$ defined on $x$-line.

e.g. "Bisection", trying to find minimum in interval $[x_0, x_1]$: Replace interval with either $[x_0; (x_0+x_1)/2]$ or $[(x_0+x_1)/2, x_1]$; make sensible choice. Iterate many times.

To reduce #iterations use smarter models of $f$: see, e.g., "secant method". Harder when $f$ varies more.

How to find minimum of function $f$ defined on $(x, y)$-plane?

"Gradient descent": Starting from $(x_0, y_0)$, try to figure out direction where $f$ decreases fastest. Could do line search to find minimum in that direction. Then find a new direction.

Better: Step down that direction. Then find a new direction.

Silly: Line search in $x$ direction; line search in $y$ direction; repeat.

Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.
You start with simple C code implementing Keccak.
How to find minimum of function \( f \) defined on \((x, y)\)-plane?

“Gradient descent”:
Starting from \((x_0, y_0)\),
try to figure out direction
where \( f \) decreases fastest.

Could do line search to find
minimum in that direction.
Then find a new direction.

Better: Step down that direction.
Then find a new direction.

Silly: Line search in \( x \) direction;
line search in \( y \) direction; repeat.

Keccak optimization

Goal: Fastest C code
on a Cortex-M4 CPU core.

You start with simple C code
implementing Keccak.
How to find minimum of function $f$ defined on $(x, y)$-plane?

"Gradient descent":
Starting from $(x_0, y_0)$, try to figure out direction where $f$ decreases fastest.

Could do line search to find minimum in that direction.
Then find a new direction.

Better: Step down that direction.
Then find a new direction.

Silly: Line search in $x$ direction; line search in $y$ direction; repeat.

Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.
How to find minimum of function \( f \) defined on \((x, y)\)-plane?

“Gradient descent”: Starting from \((x_0, y_0)\), try to figure out direction where \( f \) decreases fastest.

Could do line search to find minimum in that direction. Then find a new direction.

Better: Step down that direction. Then find a new direction.

Silly: Line search in \( x \) direction; line search in \( y \) direction; repeat.

Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.
How to find minimum of function \( f \) defined on \((x, y)\)-plane?

“Gradient descent”: Starting from \((x_0, y_0)\), try to figure out direction where \( f \) decreases fastest.

Could do line search to find minimum in that direction. Then find a new direction.

Better: Step down that direction. Then find a new direction.

Silly: Line search in \( x \) direction; line search in \( y \) direction; repeat.

Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.
How to find minimum of function $f$ defined on $(x, y)$-plane?

“Gradient descent”:
Starting from $(x_0, y_0)$, try to figure out direction where $f$ decreases fastest.

Could do line search to find minimum in that direction. Then find a new direction.

Better: Step down that direction. Then find a new direction.

Silly: Line search in $x$ direction; line search in $y$ direction; repeat.

Keccak optimization
Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.

You publish your fastest code.

Maybe lots of people use it, and care about its speed.
How to find minimum of function $f$ defined on $(x, y)$-plane?

"Gradient descent": Starting from $(x_0, y_0)$, figure out direction where $f$ decreases fastest.

Do line search to find minimum in that direction.

Step down that direction.

Then find a new direction.

Silly: Line search in $x$ direction; line search in $y$ direction; repeat.

Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.

You publish your fastest code.

Maybe lots of people use it, and care about its speed.
How to find minimum of function $f$ defined on $(x; y)$-plane?

“Gradient descent”:
Starting from $(x_0; y_0)$, try to figure out direction where $f$ decreases fastest.
Could do line search to find minimum in that direction.
Then find a new direction.
Better: Step down that direction.
Then find a new direction.
Silly: Line search in $x$ direction; line search in $y$ direction; repeat.

Keccak optimization
Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.
You start with simple C code implementing Keccak.
You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.
You publish your fastest code.
Maybe lots of people use it, and care about its speed.
How to find minimum of function $f$ defined on $(x; y)$-plane?

"Gradient descent": Starting from $(x_0; y_0)$, try to figure out direction where $f$ decreases fastest. Could do line search to find minimum in that direction. Then find a new direction. Better: Step down that direction. Then find a new direction. Silly: Line search in $x$ direction; line search in $y$ direction; repeat.

Keccak optimization
Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.

You publish your fastest code.

Maybe lots of people use it, and care about its speed.

Compiler writer learns about your Keccak Cortex-M4 C code.
Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.

You publish your fastest code.

Maybe lots of people use it, and care about its speed.
Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.

You publish your fastest code.

Maybe lots of people use it, and care about its speed.

Compiler writer learns about your Keccak Cortex-M4 C code.

Compiles it; sees how fast it is. Modifies compiler to try to make the compiled code faster. Repeats; eventually stops trying.
Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.

You publish your fastest code.

Maybe lots of people use it, and care about its speed.

Compiler writer learns about your Keccak Cortex-M4 C code.

Compiles it; sees how fast it is. Modifies compiler to try to make the compiled code faster. Repeats; eventually stops trying. Publishes a new compiler version.
Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.

You publish your fastest code.

Maybe lots of people use it, and care about its speed.
Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.

You publish your fastest code.

Maybe lots of people use it, and care about its speed.

Compiler writer learns about your Keccak Cortex-M4 C code.
Compiles it; sees how fast it is.
Modifies compiler to try to make the compiled code faster.
Repeats; eventually stops trying.
Publishes a new compiler version.
Later: Maybe you try the new compiler. Whole process repeats.
You treat compiler as constant.
Compiler treats code as constant.
Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.

Compile it; see how fast it is; try to make it faster; repeat; eventually stop trying.

You publish your fastest code. Maybe lots of people use it, and care about its speed.

Compiler writer learns about your Keccak Cortex-M4 C code.

Compiles it; sees how fast it is. Modifies compilation process to try to make the compiled code faster. Repeats; eventually stops trying. Publishes a new compiler version.

Later: Maybe you try the new compiler. Whole process repeats.

You treat compiler as constant. Compiler treats code as constant.

Define $f(x; y)$ as the time taken by code $x$ with compiler $y$.
Keccak optimization
Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.
You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.
You publish your fastest code.
Maybe lots of people use it, and care about its speed.

Compiler writer learns about your Keccak Cortex-M4 C code.
Compiles it; sees how fast it is.
Modifies compiler to try to make the compiled code faster.
Repeats; eventually stops trying.
Publishes a new compiler version.
Later: Maybe you try the new compiler. Whole process repeats.
You treat compiler as constant.
Compiler treats code as constant.

Define $f(x, y)$ as the time taken by code $x$ with compiler $y$. 


Keccak optimization

Goal: Fastest C code for Keccak on a Cortex-M4 CPU core.

You start with simple C code implementing Keccak.
You compile it; see how fast it is; modify it to try to make it faster; repeat; eventually stop trying.
You publish your fastest code.
Maybe lots of people use it, and care about its speed.

Compiler writer learns about your Keccak Cortex-M4 C code.
Compiles it; sees how fast it is.
Modifies compiler to try to make the compiled code faster.
Repeats; eventually stops trying.
Publishes a new compiler version.
Later: Maybe you try the new compiler. Whole process repeats.

You treat compiler as constant.
Compiler treats code as constant.

Define \( f(x, y) \) as time taken by code \( x \) with compiler \( y \).
Compiler writer learns about your Keccak Cortex-M4 C code.
Compiles it; sees how fast it is.
Modifies compiler to try to make the compiled code faster.
Repeats; eventually stops trying.
Publishes a new compiler version.
Later: Maybe you try the new compiler. Whole process repeats.
You treat compiler as constant.
Compiler treats code as constant.

Define $f(x, y)$ as time taken by code $x$ with compiler $y$. 
Compiler writer learns about your Keccak Cortex-M4 C code.
Compiles it; sees how fast it is. Modifies compiler to try to make the compiled code faster. Repeats; eventually stops trying.
Publishes a new compiler version.
Later: Maybe you try the new compiler. Whole process repeats.
You treat compiler as constant. Compiler treats code as constant.

Define $f(x, y)$ as time taken by code $x$ with compiler $y$.
$x_0$: initial code.
$y_0$: initial compiler.
Compiler writer learns about your Keccak Cortex-M4 C code.
Compiles it; sees how fast it is. Modifies \textit{compiler} to try to make the compiled code faster. Repeats; eventually stops trying. Publishes a new compiler version.
Later: Maybe you try the new compiler. Whole process repeats.
You treat compiler as constant. Compiler treats code as constant.

Define \( f(x, y) \) as time taken by code \( x \) with compiler \( y \).
\( x_0 \): initial code.
\( y_0 \): initial compiler.
You try to minimize \( f(x, y_0) \).
\( x_1 \): new code from this line search in \( x \) direction.
Compiler writer learns about your Keccak Cortex-M4 C code.
Compiles it; sees how fast it is. Modifies compiler to try to make the compiled code faster. Repeats; eventually stops trying. Publishes a new compiler version. Later: Maybe you try the new compiler. Whole process repeats. You treat compiler as constant. Compiler treats code as constant.

Define $f(x, y)$ as time taken by code $x$ with compiler $y$.

$x_0$: initial code.
$y_0$: initial compiler.
You try to minimize $f(x, y_0)$.

$x_1$: new code from this line search in $x$ direction.

Compiler writer: $f(x_1, y)$.
$y_1$: new compiler from this line search in $y$ direction.
Compiler writer learns about your Keccak Cortex-M4 C code. Compiles it; sees how fast it is. Modifies *compiler* to try to make the compiled code faster. Repeats; eventually stops trying. Publishes a new compiler version. Later: Maybe you try the new compiler. Whole process repeats.

You treat compiler as constant. Compiler treats code as constant.

Define $f(x, y)$ as time taken by code $x$ with compiler $y$. $x_0$: initial code. $y_0$: initial compiler. You try to minimize $f(x, y_0)$. $x_1$: new code from this line search in $x$ direction. Compiler writer: $f(x_1, y)$. $y_1$: new compiler from this line search in $y$ direction. This whole approach is silly.
Compiler writer learns about your Keccak Cortex-M4 C code. Compiles it; sees how fast it is. Uses compiler to try to make compiled code faster. Eventually stops trying. Publishes a new compiler version. Maybe you try the new compiler. Whole process repeats. You treat compiler as constant. Compiler treats code as constant.

Define \( f(x, y) \) as time taken by code \( x \) with compiler \( y \).

\( x_0 \): initial code.
\( y_0 \): initial compiler.

You try to minimize \( f(x, y_0) \).

\( x_1 \): new code from this line search in \( x \) direction.

Compiler writer: \( f(x_1, y) \).

\( y_1 \): new compiler from this line search in \( y \) direction.

This whole approach is silly.

\[ \min \{ f(x, y) \} \text{ is the time taken by fastest Keccak Cortex-M4 asm.} \]
Compiler writer learns about your Keccak Cortex-M4 C code.
Compiles it; sees how fast it is.
Modifies compiler to try to make the compiled code faster.
Repeats; eventually stops trying.
Publishes a new compiler version.
Later: Maybe you try the new compiler. Whole process repeats.
You treat compiler as constant.
Compiler treats code as constant.

Define $f(x, y)$ as time taken by code $x$ with compiler $y$.

$x_0$: initial code.
$y_0$: initial compiler.

You try to minimize $f(x, y_0)$.

$x_1$: new code from this line search in $x$ direction.

Compiler writer: $f(x_1, y)$.

$y_1$: new compiler from this line search in $y$ direction.

This whole approach is silly.

$\min\{f(x, y)\}$ is the time taken by fastest Keccak Cortex-M4 asm.
Compiler writer learns about your Keccak Cortex-M4 C code. Compiles it; sees how fast it is. Modifies compiler to try to make the compiled code faster. Repeats; eventually stops trying. Publishes a new compiler version.

Later: Maybe you try the new compiler. Whole process repeats.

You treat compiler as constant. Compiler treats code as constant.

Define $f(x, y)$ as time taken by code $x$ with compiler $y$.

$x_0$: initial code.

$y_0$: initial compiler.

You try to minimize $f(x, y_0)$.

$x_1$: new code from this line search in $x$ direction.

Compiler writer: $f(x_1, y)$.

$y_1$: new compiler from this line search in $y$ direction.

This whole approach is silly.

$\min\{f(x, y)\}$ is the time taken by fastest Keccak Cortex-M4 asm.
Define $f(x, y)$ as time taken by code $x$ with compiler $y$.

$x_0$: initial code.

$y_0$: initial compiler.

You try to minimize $f(x, y_0)$.

$x_1$: new code from this line search in $x$ direction.

Compiler writer: $f(x_1, y)$.

$y_1$: new compiler from this line search in $y$ direction.

This whole approach is silly.

$\min\{f(x, y)\}$ is the time taken by fastest Keccak Cortex-M4 asm.
Define $f(x, y)$ as time taken by code $x$ with compiler $y$.

$x_0$: initial code.

$y_0$: initial compiler.

You try to minimize $f(x, y_0)$.

$x_1$: new code from this line search in $x$ direction.

Compiler writer: $f(x_1, y)$.

$y_1$: new compiler from this line search in $y$ direction.

This whole approach is silly.

$$\min\{f(x, y)\}$$ is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between $x$-line searches, $y$-line searches is a silly way to approach this min.
Define $f(x, y)$ as time taken by code $x$ with compiler $y$.

$x_0$: initial code.

$y_0$: initial compiler.

You try to minimize $f(x, y_0)$.

$x_1$: new code from this line search in $x$ direction.

Compiler writer: $f(x_1, y)$.

$y_1$: new compiler from this line search in $y$ direction.

This whole approach is silly.

$\min\{f(x, y)\}$ is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between $x$-line searches, $y$-line searches is a silly way to approach this min.

Clearly $\min$ can be achieved by many different pairs $(x, y)$.
Which pair is easiest to find?
Define $f(x, y)$ as time taken by code $x$ with compiler $y$.

$x_0$: initial code.

$y_0$: initial compiler.

You try to minimize $f(x, y_0)$.

$x_1$: new code from this line search in $x$ direction.

Compiler writer: $f(x_1, y)$.

$y_1$: new compiler from this line search in $y$ direction.

This whole approach is silly.

\[ \min \{ f(x, y) \} \] is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between $x$-line searches, $y$-line searches is a silly way to approach this min.

Clearly min can be achieved by many different pairs $(x, y)$.

Which pair is easiest to find?

Generalize from C to other languages: which language makes min easiest to find?

Why did goal say “C code”? End user doesn’t need C.
Define $f(x, y)$ as time taken by code $x$ with compiler $y$.

$x_0$: initial code.

$y_0$: initial compiler.

You try to minimize $f(x, y_0)$.

$x_1$: new code from this line search in $x$ direction.

Compiler writer: $f(x_1, y)$.

$y_1$: new compiler from this line search in $y$ direction.

This whole approach is silly.

$$\min \{ f(x, y) \}$$ is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between $x$-line searches, $y$-line searches is a silly way to approach this min.

Clearly min can be achieved by many different pairs $(x, y)$.

Which pair is easiest to find?

Generalize from C to other languages: which language makes min easiest to find?

Why did goal say “C code”?

End user doesn’t need C.

Does end user need Cortex-M4?
Define \( f(x; y) \) as time taken by code \( x \) with compiler \( y \).

\( x_0 \): initial code.
\( y_0 \): initial compiler.

You try to minimize \( f(x_0; y) \).

\( x_1 \): new code from this line search in \( x \) direction.

Compiler writer: \( f(x_1; y) \).

\( y_1 \): new compiler from this line search in \( y \) direction.

This whole approach is silly.

\[ \min \{ f(x, y) \} \] is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between \( x \)-line searches, \( y \)-line searches is a silly way to approach this min.

Clearly \( \min \) can be achieved by many different pairs \( (x, y) \).

Which pair is easiest to find?

Generalize from C to other languages: which language makes \( \min \) easiest to find?

Why did goal say “C code”?

End user doesn’t need C.

Does end user need Cortex-M4?
Define \( f(x; y) \) as time taken by code \( x \) with compiler \( y \).

\( x_0 \): initial code.
\( y_0 \): initial compiler.

You try to minimize \( f(x; y_0) \).

\( x_1 \): new code from this line search in \( x \)-direction.

Compiler writer: \( f(x_1; y) \).

\( y_1 \): new compiler from this line search in \( y \)-direction.

This whole approach is silly.

\[
\min \{ f(x, y) \}
\]
is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between \( x \)-line searches, \( y \)-line searches is a silly way to approach this min.

Clearly \( \min \) can be achieved by many different pairs \((x, y)\).

Which pair is easiest to find?

Generalize from C to other languages: which language makes \( \min \) easiest to find?

Why did goal say “C code”? End user doesn’t need C.

Does end user need Cortex-M4?
min\{f(x, y)\} is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between \(x\)-line searches, \(y\)-line searches is a silly way to approach this min.

Clearly min can be achieved by many different pairs \((x, y)\).
Which pair is easiest to find?

Generalize from C to other languages: which language makes min easiest to find?
Why did goal say “C code”? End user doesn’t need C.

Does end user need Cortex-M4?
\( \min\{f(x, y)\} \) is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between \( x \)-line searches, \( y \)-line searches is a silly way to approach this \( \min \).

Clearly \( \min \) can be achieved by many different pairs \( (x, y) \).
Which pair is easiest to find?

Generalize from C to other languages: which language makes \( \min \) easiest to find?

Why did goal say “C code”?
End user doesn’t need C.

Does end user need Cortex-M4?
CPU designer learns about your Keccak Cortex-M4 asm.
min\{f(x, y)\} \text{ is the time taken by fastest Keccak Cortex-M4 asm.}

Slowly bouncing between x-line searches, y-line searches is a silly way to approach this min.

Clearly min can be achieved by many different pairs $$(x, y)$$.
Which pair is easiest to find?

Generalize from C to other languages: which language makes min easiest to find?
Why did goal say “C code”? End user doesn’t need C.
\[ \min \{ f(x, y) \} \] is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between \( x \)-line searches, \( y \)-line searches is a silly way to approach this min.

Clearly \( \min \) can be achieved by many different pairs \((x, y)\). Which pair is easiest to find?

Generalize from C to other languages: which language makes \( \min \) easiest to find?

Why did goal say “C code”? End user doesn’t need C.

Does end user need Cortex-M4?

CPU designer learns about your Keccak Cortex-M4 asm.

Modifies the CPU design to try to make this code faster. Repeats; eventually stops trying.

Years later, sells a new CPU. You reoptimize for this CPU.
$$\min \{ f(x, y) \}$$ is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between $x$-line searches, $y$-line searches is a silly way to approach this min.

Clearly min can be achieved by many different pairs ($x, y$). Which pair is easiest to find?

Generalize from C to other languages: which language makes min easiest to find? Why did goal say “C code”? End user doesn’t need C.

Does end user need Cortex-M4? CPU designer learns about your Keccak Cortex-M4 asm.

Modifies the CPU design to try to make this code faster. Repeats; eventually stops trying.

Years later, sells a new CPU. You reoptimize for this CPU.

*Sometimes* CPUs try extending or replacing instruction set, but this is poorly coordinated with programmers, compiler writers.
\( f(x, y) \) is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between \( x \)-line searches, \( y \)-line searches is a silly way to approach this min. Clearly min can be achieved by different pairs \((x, y)\).

Which pair is easiest to find?

Generalize from C to other languages: which language makes min easiest to find?

Why did goal say “C code”? End user doesn’t need C.

Does end user need Cortex-M4?

CPU designer learns about your Keccak Cortex-M4 asm.

Modifies the CPU design to try to make this code faster. Repeats; eventually stops trying.

Years later, sells a new CPU. You reoptimize for this CPU.

Sometimes CPUs try extending or replacing instruction set, but this is poorly coordinated with programmers, compiler writers.

Generalize \( f(x, y) \): given code \( x \) on platform \( y \),

If compiler on code \( x \) produces asm \( y(x) \) for Cortex-M4:

\[
f(x, y) = f(y(x); \text{Cortex-M4}).
\]
min \{ f(x; y) \} is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between x-line searches, y-line searches is a silly way to approach this min.

Clearly min can be achieved by many different pairs \((x; y)\).

Which pair is easiest to find?

Generalize from C to other languages: which language makes min easiest to find?

Why did goal say “C code”?

End user doesn’t need C.

Does end user need Cortex-M4?

CPU designer learns about your Keccak Cortex-M4 asm.

Modifies the CPU design to try to make this code faster.

Repeats; eventually stops trying.

Years later, sells a new CPU.

You reoptimize for this CPU.

Sometimes CPUs try extending or replacing instruction set, but this is poorly coordinated with programmers, compiler writers.

Generalize \(f(x, y)\) equation:

\(f(x, y)\) is time taken by code \(x\) on platform \(y\).

If compiler \(y\) on code \(x\) produces \(\text{asm } y(x)\) for Cortex-M4:

\(f(x, y) = f(y(x), \text{Cortex-M4})\).
min \{ f(x; y) \} is the time taken by fastest Keccak Cortex-M4 asm.

Slowly bouncing between \( x \)-line searches, \( y \)-line searches is a silly way to approach this min.

Clearly min can be achieved by many different pairs \((x; y)\).

Which pair is easiest to find?

Generalize from C to other languages: which language makes min easiest to find?

Why did goal say "C code"?
End user doesn't need C.

Does end user need Cortex-M4?
CPU designer learns about your Keccak Cortex-M4 asm.

Modifies the CPU design to try to make this code faster.
Repeats; eventually stops trying.

Years later, sells a new CPU.
You reoptimize for this CPU.

Sometimes CPUs try extending or replacing instruction set, but this is poorly coordinated with programmers, compiler writers.

Generalize \( f(x, y) \) definition:
\( f(x, y) \) is time taken by code \( x \) on platform \( y \).

If compiler \( y \) on code \( x \) produces \( \text{asm } y(x) \) for Cortex-M4:
\( f(x, y) = f(y(x), \text{Cortex-M4}) \).
Does end user need Cortex-M4?

CPU designer learns about your Keccak Cortex-M4 asm.

Modifies the CPU design to try to make this code faster.
Repeats; eventually stops trying.

Years later, sells a new CPU.
You reoptimize for this CPU.

Sometimes CPUs try extending or replacing instruction set, but this is poorly coordinated with programmers, compiler writers.

Generalize $f(x, y)$ definition:

$f(x, y)$ is time taken by code $x$ on platform $y$.

If compiler $y$ on code $x$ produces asm $y(x)$ for Cortex-M4:

$f(x, y) = f(y(x), \text{Cortex-M4})$. 
Does end user need Cortex-M4?
CPU designer learns about your Keccak Cortex-M4 asm.
Modifies the CPU design to try to make this code faster.
Repeats; eventually stops trying.
Years later, sells a new CPU.
You reoptimize for this CPU.
Sometimes CPUs try extending or replacing instruction set, but this is poorly coordinated with programmers, compiler writers.

Generalize $f(x, y)$ definition:
$f(x, y)$ is time taken by code $x$ on platform $y$.
If compiler $y$ on code $x$ produces asm $y(x)$ for Cortex-M4:
$f(x, y) = f(y(x), \text{Cortex-M4})$.
Without the CPU changing:
Minimize $f(a, \text{Cortex-M4})$.
Search for $(x, y)$ with $y(x) = a$. 
Does end user need Cortex-M4?
CPU designer learns about your Keccak Cortex-M4 asm.
Modifies the CPU design to try to make this code faster.
Repeats; eventually stops trying.
Years later, sells a new CPU.
You reoptimize for this CPU.

_Sometimes_ CPUs try extending or replacing instruction set, but this is poorly coordinated with programmers, compiler writers.

Generalize $f(x, y)$ definition:

$f(x, y)$ is time taken by code $x$ on platform $y$.

If compiler $y$ on code $x$ produces $y(x)$ for Cortex-M4:

$f(x, y) = f(y(x), \text{Cortex-M4})$.

Without the CPU changing:
Minimize $f(a, \text{Cortex-M4})$.
Search for $(x, y)$ with $y(x) = a$.

Typical CPU designer:
View $a$ as a constant; try to minimize $f(a, y)$.
Silly optimization approach.
Does end user need Cortex-M4? CPU designer learns about your Keccak Cortex-M4 asm.
Modifies the CPU design to make this code faster. Eventually stops trying.
Sells a new CPU. You reoptimize for this CPU.

Sometimes CPUs try extending or replacing instruction set, but this is poorly coordinated with programmers, compiler writers.

Generalize $f(x, y)$ definition: $f(x, y)$ is time taken by code $x$ on platform $y$.

If compiler $y$ on code $x$ produces asm $y(x)$ for Cortex-M4: $f(x, y) = f(y(x), \text{Cortex-M4})$.

Without the CPU changing:
Minimize $f(a, \text{Cortex-M4})$.
Search for $(x, y)$ with $y(x) = a$.

Typical CPU designer:
View $a$ as a constant; try to minimize $f(a, y)$.
Silly optimization approach.

“I know the minimum! I’ve developed the fastest circuit that computes Keccak. This circuit is my CPU.”
Does end user need Cortex-M4?

CPU designer learns about your Keccak Cortex-M4 asm.

Modifies the CPU design to try to make this code faster.

Eventually stops trying.

Years later, sells a new CPU.

You reoptimize for this CPU.

Sometimes CPUs try extending or replacing instruction set, but this is poorly coordinated with programmers, compiler writers.

Generalize $f(x, y)$ definition:

$f(x, y)$ is time taken by code $x$ on platform $y$.

If compiler $y$ on code $x$ produces asm $y(x)$ for Cortex-M4:

$f(x, y) = f(y(x), \text{Cortex-M4})$.

Without the CPU changing:

Minimize $f(a, \text{Cortex-M4})$.

Search for $(x, y)$ with $y(x) = a$.

Typical CPU designer:

View $a$ as a constant; try to minimize $f(a, y)$.

Silly optimization approach.

“I know the minimum! I’ve developed the fastest circuit that computes Keccak. This circuit is my CPU.”
Generalize $f(x, y)$ definition:
$f(x, y)$ is time taken by code $x$ on platform $y$.

If compiler $y$ on code $x$ produces
asm $y(x)$ for Cortex-M4:
$f(x, y) = f(y(x), $Cortex-M4$)$.

Without the CPU changing:
Minimize $f(a, $Cortex-M4$)$.
Search for $(x, y)$ with $y(x) = a$.

Typical CPU designer:
View $a$ as a constant;
try to minimize $f(a, y)$.
Silly optimization approach.

“I know the minimum!
I’ve developed the fastest circuit
that computes Keccak.
This circuit is my CPU.”
Generalize $f(x, y)$ definition: $f(x, y)$ is time taken by code $x$ on platform $y$.

If compiler $y$ on code $x$ produces asm $y(x)$ for Cortex-M4: $f(x, y) = f(y(x), \text{Cortex-M4})$.

Without the CPU changing:
Minimize $f(a, \text{Cortex-M4})$.
Search for $(x, y)$ with $y(x) = a$.

Typical CPU designer:
View $a$ as a constant;
try to minimize $f(a, y)$.
Silly optimization approach.

“I know the minimum!
I’ve developed the fastest circuit that computes Keccak.
This circuit is my CPU.”
Generalize $f(x, y)$ definition: $f(x, y)$ is time taken by code $x$ on platform $y$.

If compiler $y$ on code $x$ produces $\text{asm } y(x)$ for Cortex-M4:

$$f(x, y) = f(y(x), \text{Cortex-M4}).$$

Without the CPU changing:

Minimize $f(a, \text{Cortex-M4})$.

Search for $(x, y)$ with $y(x) = a$.

Typical CPU designer:
View $a$ as a constant;
try to minimize $f(a, y)$.

Silly optimization approach.

“I know the minimum!
I’ve developed the fastest circuit
that computes Keccak.
This circuit is my CPU.”

Wait a minute: “CPU” concept is more restrictive than “chip”.

Perspective of CPU designer:
This chip can do anything!

People want this chip to support SHA-1, SHA-2, SHA-3, SHAmir;
all sorts of block ciphers;
public-key cryptosystems;
non-cryptographic computations.
Generalize $f(x, y)$ definition: $f(x, y)$ is time taken by code $x$ on platform $y$. If compiler $y$ on code $x$ produces $asm_y(x)$ for Cortex-M4: $f(x, y) = f(y(x), \text{Cortex-M4})$. Without the CPU changing: $f(a, \text{Cortex-M4})$. For $(x, y)$ with $y(x) = a$. Typical CPU designer: View $a$ as a constant; try to minimize $f(a, y)$. Silly optimization approach.

“I know the minimum! I’ve developed the fastest circuit that computes Keccak. This circuit is my CPU.”

Wait a minute: “CPU” concept is more restrictive than “chip”.

Perspective of CPU designer: This chip can do anything!

People want this chip to support SHA-1, SHA-2, SHA-3, SHAmir; all sorts of block ciphers; public-key cryptosystems; non-cryptographic computations.

Adding fast Keccak circuit (“Keccak coprocessor”) to CPU adds area to CPU. Adding fast coprocessors for desired mix of operations adds even more area to CPU.
Generalize $f(x; y)$ definition: $f(x; y)$ is time taken by code $x$ on platform $y$. 

If compiler $y$ on code $x$ produces $asm_y(x)$ for Cortex-M4: 

$$f(x; y) = f(y(x); \text{Cortex-M4}).$$

Without changing: 

Minimize $f(a; \text{Cortex-M4}).$

Search for $(x; y)$ with $y(x) = a$.

Typical CPU designer: 

View $a$ as a constant; try to minimize $f(a; y)$.

Silly optimization approach.

“I know the minimum! I’ve developed the fastest circuit that computes Keccak. This circuit is my CPU.”

Wait a minute: “CPU” concept is more restrictive than “chip”.

Perspective of CPU designer: 

This chip can do anything!

People want this chip to support SHA-1, SHA-2, SHA-3, SHAmir; all sorts of block ciphers; public-key cryptosystems; non-cryptographic computations.

Adding fast Keccak circuit (“Keccak coprocessor”) to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area.
Generalize \( f(x; y) \) definition:
\( f(x; y) \) is time taken by code \( x \) on platform \( y \).
If compiler \( y \) on code \( x \) produces \( \text{asm} \)(\( x \)) for Cortex-M4:
\[ f(x; y) = f(y(x)) \text{; Cortex-M4} \]

Without the CPU changing:
Minimize \( f(a; \text{Cortex-M4}) \).
Search for \((x; y)\) with \( y(x) = a \).

Typical CPU designer:
View \( a \) as a constant;
try to minimize \( f(a; y) \).
Silly optimization approach.

“I know the minimum!
I’ve developed the fastest circuit that computes Keccak.
This circuit is my CPU.”

Wait a minute: “CPU” concept is more restrictive than “chip”.

Perspective of CPU designer:
This chip can do anything!

People want this chip to support SHA-1, SHA-2, SHA-3, SHAMir;
all sorts of block ciphers;
public-key cryptosystems;
non-cryptographic computations.

Adding fast Keccak circuit (“Keccak coprocessor”) to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area to CPU.
“I know the minimum! I’ve developed the fastest circuit that computes Keccak. This circuit is my CPU.”

Wait a minute: “CPU” concept is more restrictive than “chip”.

Perspective of CPU designer: This chip can do anything!

People want this chip to support SHA-1, SHA-2, SHA-3, SHAmir; all sorts of block ciphers; public-key cryptosystems; non-cryptographic computations.

Adding fast Keccak circuit (“Keccak coprocessor”) to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area to CPU.
“I know the minimum! I’ve developed the fastest circuit that computes Keccak. This circuit is my CPU.”

Wait a minute: “CPU” concept is more restrictive than “chip”.

Perspective of CPU designer: This chip can do anything!

People want this chip to support SHA-1, SHA-2, SHA-3, SHAmir; all sorts of block ciphers; public-key cryptosystems; non-cryptographic computations.

Adding fast Keccak circuit (“Keccak coprocessor”) to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area to CPU.

For same CPU area, obtain much better throughput by building many copies of original CPU core without these coprocessors.
“I know the minimum! I’ve developed the fastest circuit that computes Keccak. This circuit is my CPU.”

Wait a minute: “CPU” concept is more restrictive than “chip”.

Perspective of CPU designer: This chip can do anything!

People want this chip to support SHA-1, SHA-2, SHA-3, SHAaMIR; all sorts of block ciphers; public-key cryptosystems; non-cryptographic computations.

Adding fast Keccak circuit (“Keccak coprocessor”) to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area to CPU.

For same CPU area, obtain much better throughput by building many copies of original CPU core without these coprocessors.

Fast Keccak chip is special case. Doesn’t reflect general case.
"I know the minimum! I've developed the fastest circuit that computes Keccak. This circuit is my CPU."

Wait a minute: "CPU" concept is more restrictive than "chip".

Perspective of CPU designer: This chip can do anything! People want this chip to support SHA-2, SHA-3, SHAmir; all sorts of block ciphers; public-key cryptosystems; non-cryptographic computations.

Adding fast Keccak circuit ("Keccak coprocessor") to CPU adds area to CPU. Adding fast coprocessors for desired mix of operations adds even more area to CPU.

For same CPU area, obtain much better throughput by building many copies of original CPU core without these coprocessors. Fast Keccak chip is special case. Doesn’t reflect general case.

CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?
I know the minimum!
I’ve developed the fastest circuit
that computes Keccak.
This circuit is my CPU.

Wait a minute: “CPU” concept
is more restrictive than “chip”.

Perspective of CPU designer:
This chip can do anything!
People want this chip to support
SHA-1, SHA-2, SHA-3, SHA256;
all sorts of block ciphers;
public-key cryptosystems;
non-cryptographic computations.

Adding fast Keccak circuit
(“Keccak coprocessor”) to CPU
adds area to CPU.

Adding fast coprocessors
for desired mix of operations
adds even more area to CPU.

For same CPU area,
obtain much better throughput
by building many copies
of original CPU core
without these coprocessors.

Fast Keccak chip is special case.
Doesn’t reflect general case.

CPU designer’s metric:
What is best performance
for a specified mix of operations
within a particular CPU area?
Adding fast Keccak circuit ("Keccak coprocessor") to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area to CPU.

For same CPU area, obtain much better throughput by building many copies of original CPU core without these coprocessors.

Fast Keccak chip is special case. Doesn’t reflect general case.

CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?
Adding fast Keccak circuit ("Keccak coprocessor") to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area to CPU.

For same CPU area, obtain much better throughput by building many copies of original CPU core without these coprocessors.

Fast Keccak chip is special case. Doesn’t reflect general case.

CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?
Adding fast Keccak circuit ("Keccak coprocessor") to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area to CPU.

For same CPU area, obtain much better throughput by building many copies of original CPU core without these coprocessors.

Fast Keccak chip is special case. Doesn’t reflect general case.

CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?

CPU designer is much more likely to consider incorporating a small Keccak coprocessor.
Adding fast Keccak circuit ("Keccak coprocessor") to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area to CPU.

For same CPU area, obtain much better throughput by building many copies of original CPU core without these coprocessors.

Fast Keccak chip is special case. Doesn’t reflect general case.

CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?

CPU designer is much more likely to consider incorporating a **small** Keccak coprocessor.

“So we should design the **smallest** Keccak circuit?”
Adding fast Keccak circuit ("Keccak coprocessor") to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations adds even more area to CPU.

For same CPU area, obtain much better throughput by building many copies of original CPU core without these coprocessors.

Fast Keccak chip is special case. Doesn’t reflect general case.

CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?

CPU designer is much more likely to consider incorporating a small Keccak coprocessor.

“So we should design the smallest Keccak circuit?”

—Maybe, but will this extreme be faster than using existing CPU instructions without coprocessor?
Adding fast Keccak circuit (“Keccak coprocessor”) to CPU adds area to CPU.

Adding fast coprocessors for desired mix of operations even more area to CPU.

For the same CPU area, much better throughput by building many copies of original CPU core without these coprocessors.

Fast Keccak chip is special case. Doesn’t reflect general case.

CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?

CPU designer is much more likely to consider incorporating a small Keccak coprocessor.

“So we should design the smallest Keccak circuit?”

—Maybe, but will this extreme be faster than using existing CPU instructions without coprocessor?

Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

“So it’s a small cost for Intel to add instruction-set extension for my favorite crypto!”
Adding fast Keccak circuit ("Keccak coprocessor") to CPU area to CPU.

Adding fast coprocessors for desired mix of operations adds area to CPU.

For the same CPU area, obtain much better throughput by building many copies of the original CPU core without these coprocessors.

Fast Keccak chip is a special case. It doesn’t reflect the general case.

CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?

CPU designer is much more likely to consider incorporating a small Keccak coprocessor.

“So we should design the smallest Keccak circuit?”

—Maybe, but will this extreme be faster than using existing CPU instructions without coprocessor?

Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

“So it’s small cost for Intel to add instruction-set extension for my favorite crypto!”
Adding fast Keccak circuit ("Keccak coprocessor") to CPU adds area to CPU. Adding fast coprocessors for desired mix of operations adds even more area to CPU. For same CPU area, obtain much better throughput by building many copies of original CPU core without these coprocessors. Fast Keccak chip is special case. Doesn’t reflect general case.

CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area? CPU designer is much more likely to consider incorporating a small Keccak coprocessor. “So we should design the smallest Keccak circuit?” —Maybe, but will this extreme be faster than using existing CPU instructions without coprocessor?

Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc. “So it’s small cost for Intel to add instruction-set extension for my favorite crypto!”
CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?

CPU designer is much more likely to consider incorporating a small Keccak coprocessor.

“So we should design the smallest Keccak circuit?”

—Maybe, but will this extreme be faster than using existing CPU instructions without coprocessor?

Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

“So it’s small cost for Intel to add instruction-set extension for my favorite crypto!”
CPU designer’s metric: What is best performance for a specified mix of operations within a particular CPU area?

CPU designer is much more likely to consider incorporating a small Keccak coprocessor.

“So we should design the smallest Keccak circuit?”

—Maybe, but will this extreme be faster than using existing CPU instructions without coprocessor?

Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

“So it’s small cost for Intel to add instruction-set extension for my favorite crypto!”

—Yes, but even smaller benefit for Intel’s mix of operations.
CPU designer’s metric:

What is best performance for a specified mix of operations within a particular CPU area?

CPU designer is much more likely to consider incorporating a small Keccak coprocessor.

“So we should design the smallest Keccak circuit?”

—Maybe, but will this extreme be faster than using existing CPU instructions without coprocessor?

Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

“So it’s small cost for Intel to add instruction-set extension for my favorite crypto!”

—Yes, but even smaller benefit for Intel’s mix of operations.

Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
... 1: probably not worthwhile compared to skipping coprocessor and using existing CPU instructions.
CPU designer's metric: What is best performance for a specified mix of operations within a particular CPU area? Much more likely to consider incorporating a small Keccak coprocessor.

“So we should design the smallest Keccak circuit?” — Maybe, but will this extreme be faster than using existing CPU instructions without coprocessor?

Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

“So it's small cost for Intel to add instruction-set extension for my favorite crypto!”

—Yes, but even smaller benefit for Intel’s mix of operations.

Intel did add instruction for 1 round of AES. How many parallel S-boxes are in an AES-round coprocessor? Can be 16: big; fast. 8: smaller but slower. 4: even smaller but still fast. … 1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.
CPU designer's metric: What is best performance for a specified mix of operations within a particular CPU area? CPU designer is much more likely to consider incorporating a small Keccak coprocessor. 

"So we should design the smallest Keccak circuit?" — Maybe, but will this extreme be faster than using existing CPU instructions without coprocessor?

Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

“So it’s small cost for Intel to add instruction-set extension for my favorite crypto!”

—Yes, but even smaller benefit for Intel’s mix of operations.

Intel did add instruction for 1 round of AES. How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
... 1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.
Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

“So it’s small cost for Intel to add instruction-set extension for my favorite crypto!”

—Yes, but even smaller benefit for Intel’s mix of operations.

Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
... 1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.
Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

“So it’s small cost for Intel to add instruction-set extension for my favorite crypto!”

—Yes, but even smaller benefit for Intel’s mix of operations.

Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.

... 1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.

An instruction for 4 rounds of SHA-256 is in a few Intel CPUs.
Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc. “So it’s small cost for Intel to add instruction-set extension for my favorite crypto!” —Yes, but even smaller benefit for Intel’s mix of operations.

Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
... 1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.

An instruction for 4 rounds of SHA-256 is in a few Intel CPUs.

Lightweight crypto
Frequent claim in literature, where $X$ might be
• Keccak;
• any secure hash;
• a secure cipher;
... “Resource-constrained IoT devices need the smallest circuit for $X$. 

Intel typically designs quite large CPU cores: 32KB L1 data cache, 32KB L1 instruction cache, several fast multipliers, many different instructions, out-of-order unit, etc.

"So it's small cost for Intel to add instruction-set extension for my favorite crypto!"

—Yes, but even smaller benefit for Intel's mix of operations.

Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
... 1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.

An instruction for 4 rounds of SHA-256 is in a few Intel CPUs.

Lightweight crypto

Frequent claim in literature, where $X$ might be:
- Keccak;
- any secure hash;
- a secure cipher; ... :

"Resource-constrained IoT devices need the smallest circuit for $X$."
Intel typically designs quite large CPU cores:
32KB L1 data cache,
32KB L1 instruction cache,
several fast multipliers,
many different instructions,
out-of-order unit, etc.

"So it's small cost for Intel to add instruction-set extension for my favorite crypto!"
—Yes, but even smaller benefit for Intel's mix of operations.

Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?
Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
... 1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.

An instruction for 4 rounds of SHA-256 is in a few Intel CPUs.

Lightweight crypto
Frequent claim in literature, where X might be
• Keccak;
• any secure hash;
• a secure cipher; . . . :
“Resource-constrained IoT devices need the smallest circuit for X ."
Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
... 1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.

An instruction for 4 rounds of SHA-256 is in a few Intel CPUs.

Lightweight crypto

Frequent claim in literature, where $X$ might be

- Keccak;
- any secure hash;
- a secure cipher;

“Resource-constrained IoT devices need the smallest circuit for $X$.”
Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.

An instruction for 4 rounds of SHA-256 is in a few Intel CPUs.

Lightweight crypto
Frequent claim in literature, where $X$ might be
• Keccak;
• any secure hash;
• a secure cipher; ...

“Resource-constrained IoT devices need the smallest circuit for $X$.”

—Even if speed is acceptable, who will use smallest $X$ circuit?
Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.

An instruction for 4 rounds of SHA-256 is in a few Intel CPUs.

Lightweight crypto
Frequent claim in literature, where $X$ might be
• Keccak;
• any secure hash;
• a secure cipher; 

“Resource-constrained IoT devices need the smallest circuit for $X$."

—Even if speed is acceptable, who will use smallest $X$ circuit?

Why should minimum area for $X$ give minimum area for IoT+$X$?
Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

16: big; fast.
8: smaller but slower.
4: even smaller but slower.
1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.

An instruction for 4 rounds of SHA-256 is in a few Intel CPUs.

Lightweight crypto
Frequent claim in literature, where $X$ might be
- Keccak;
- any secure hash;
- a secure cipher; . . . :

“Resource-constrained IoT devices need the smallest circuit for $X$.”

—Even if speed is acceptable, who will use smallest $X$ circuit?

Why should minimum area for $X$ give minimum area for IoT+ $X$?

An idea from Adam Langley
Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

E.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.
Intel did add instruction for 1 round of AES.

How many parallel S-boxes are in an AES-round coprocessor?

Can be 16: big; fast.
8: smaller but slower.
4: even smaller but slower.
1: probably not worthwhile compared to skipping coprocessor and using other CPU instructions.

An instruction for 4 rounds of SHA-256 is in a few Intel CPUs.

Lightweight crypto

Frequent claim in literature, where X might be
- Keccak;
- any secure hash;
- a secure cipher; . . . :

“Resource-constrained IoT devices need the smallest circuit for X.”

—Even if speed is acceptable, who will use smallest X circuit?

Why should minimum area for X give minimum area for IoT+X?

An idea from Adam Langley

Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

E.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.
Lightweight crypto

Frequent claim in literature, where X might be
- Keccak;
- any secure hash;
- a secure cipher; . . . :

“Resource-constrained IoT devices need the smallest circuit for X.”

—Even if speed is acceptable, who will use smallest X circuit?

Why should minimum area for X give minimum area for IoT+X?

An idea from Adam Langley

Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

e.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions, since old functions were broken.
Lightweight crypto

Frequent claim in literature, where $X$ might be

- Keccak;
- any secure hash;
- a secure cipher; . . .:

"Resource-constrained IoT devices need the smallest circuit for $X$.”

—Even if speed is acceptable, who will use smallest $X$ circuit?

Why should minimum area for $X$ give minimum area for IoT$+X$?

An idea from Adam Langley

Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

e.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.
Lightweight crypto

Frequent claim in literature, where $X$ might be
• Keccak;
• any secure hash;
• a secure cipher; . . . :

"Resource-constrained IoT devices need the smallest circuit for $X$.”

If speed is acceptable, who will use smallest $X$ circuit?
Why should minimum area for $X$ give minimum area for IoT+$X$?

An idea from Adam Langley

Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

E.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.

A public key is a signature-verification program in a limited language.

Langley’s idea: Replace this language with a full programming language. Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.
Lightweight crypto

Frequent claim in literature, where $X$ might be

- Keccak;
- any secure hash;
- a secure cipher;

"Resource-constrained IoT devices need the smallest circuit for $X$.”

Even if speed is acceptable, who will use smallest $X$ circuit?

Why should minimum area for $X$ give minimum area for IoT+$X$?

An idea from Adam Langley

Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

e.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.

A public key is a signature-verification program in a limited language.

Langley’s idea: Replace this language with a full programming language. Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.
An idea from Adam Langley

Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

e.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.

Langley’s idea:
Replace this language with a full programming language.
Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.
An idea from Adam Langley
Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.
e.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.

A public key is a signature-verification program in a limited language.

Langley’s idea: Replace this language with a full programming language. Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.
An idea from Adam Langley
Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

e.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.

A public key is a signature-verification program in a limited language.

Langley’s idea: Replace this language with a full programming language. Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.

Same for public-key encryption systems: public key is program.
An idea from Adam Langley

Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.

A public key is a signature-verification program in a limited language.

Langley’s idea:
Replace this language with a full programming language.

Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.

Same for public-key encryption systems: public key is program.

Say verification device is a chip of area $A$.

How small can public keys be?

Have to consider, e.g., size of a SHA-256 program, size of a Keccak program, etc.
An idea from Adam Langley

Consider a device that receives public keys from trusted sources; receives data supposedly signed under these public keys; verifies these signatures.

e.g. an SSL client.

Painful historical event: all clients needed upgrades to support new hash functions since old functions were broken.

A public key is a signature-verification program in a limited language.

Langley’s idea: Replace this language with a full programming language. Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.

Same for public-key encryption systems: public key is program.

Say verification device is a chip of area $A$. How small can public keys be?

Have to consider, e.g., size of a SHA-256 program, size of a Keccak program...
A public key is a signature-verification program in a limited language.

Langley’s idea:
Replace this language with a full programming language.
Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.

Same for public-key encryption systems: public key is program.

Say verification device is a chip of area $A$.
How small can public keys be?

Have to consider, e.g., size of a SHA-256 program, size of a Keccak program, etc.
A public key is a signature-verification program in a limited language.

Langley’s idea:
Replace this language with a full programming language. Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.

Same for public-key encryption systems: public key is program.

Say verification device is a chip of area $A$. How small can public keys be?

Have to consider, e.g., size of a SHA-256 program, size of a Keccak program, etc.
A public key is a signature-verification program in a limited language.

Langley’s idea: Replace this language with a full programming language. Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.

Same for public-key encryption systems: public key is program.

Say verification device is a chip of area $A$. How small can public keys be?

Have to consider, e.g., size of a SHA-256 program, size of a Keccak program, etc.

Similar question to optimizing total size of a CPU with a SHA-256 instruction, a Keccak instruction, etc.
A public key is a signature-verification program in a limited language.

Langley's idea:
Replace this language with a full programming language.
Then can upgrade hash function (or upgrade to post-quantum signatures!) by changing public keys, with no changes to clients.

Same for public-key encryption systems: public key is program.

Say verification device is a chip of area $A$.
How small can public keys be?

Have to consider, e.g., size of a SHA-256 program, size of a Keccak program, etc.

Similar question to optimizing total size of a CPU with a SHA-256 instruction, a Keccak instruction, etc.

Not the usual code-size question. Change the language!