Benchmarking benchmarking, and optimizing optimization

Daniel J. Bernstein

University of Illinois at Chicago & Technische Universiteit Eindhoven

Bit operations pe (assuming precon as listed in recent				
key	ops/bit	cipł		
128 128 128	88 100 117	Sim NO Skii		
256 128 256 128 128 256	144 147.2 156 162.75 202.5 283.5	Sim PR Skin Pice AES		

1

# er bit of plaintext nputed subkeys), t Skinny paper:

2

her

# non: 60 ops broken EKEON

nny

non: 106 ops broken ESENT nny colo S

Benchmarking benchmarking, and optimizing optimization

Daniel J. Bernstein

University of Illinois at Chicago & Technische Universiteit Eindhoven

Bit operations per bit of plaintext (assuming precomputed subkeys),

1

key	ops/bit	ciph
256	54	Salsa
256	78	Salsa
128	88	Simo
128	100	NOE
128	117	Skin
256	126	Salsa
256	144	Simo
128	147.2	PRE
256	156	Skin
128	162.75	Picc
128	202.5	AES
256	283.5	AES

# not entirely listed in Skinny paper:

2

### her

- sa20/8
- |sa20/12|
- non: 60 ops broken DEKEON

### nny

- sa20
- non: 106 ops broken ESENT

# nny

- colo
- S

arking benchmarking, mizing optimization 1

. Bernstein

ty of Illinois at Chicago & the Universiteit Eindhoven Bit operations per bit of plaintext (assuming precomputed subkeys), not entirely listed in Skinny paper:

l.	ana /hit	
кеу	ops/ bit	cipner
256	54	Salsa20/8
256	78	Salsa20/12
128	88	Simon: 60 op
128	100	NOEKEON
128	117	Skinny
256	126	Salsa20
256	144	Simon: 106 c
128	147.2	PRESENT
256	156	Skinny
128	162.75	Piccolo
128	202.5	AES
256	283.5	AES

### os broken

2

### ops broken

Operatio poor mo worse m Pick a c How fas First ste Write si e.g. Ber Janssen-Smetser includes impleme

# nchmarking, timization

1

n

is at Chicago & siteit Eindhoven Bit operations per bit of plaintext (assuming precomputed subkeys), not entirely listed in Skinny paper:

n
en
<b>)</b>

# Operation counts poor model of har worse model of so Pick a cipher: e.g

2

How fast is Salsa2

First step in analy Write simple softw

e.g. Bernstein-var Janssen-Lange-Sc Smetsers "TweetN includes essentially implementation of

7	
5	7

### ago & hoven

Bit operations per bit of plaintext (assuming precomputed subkeys), not entirely listed in Skinny paper:

Pick a cipher: e.g., Salsa20. ops/bit cipher key How fast is Salsa20 software 256 Salsa20/8 54 Salsa20/12 256 78 First step in analysis: Simon: 60 ops broken 128 88 Write simple software. NOEKEON 128 100 128 117 Skinny e.g. Bernstein-van Gastel-256 126 Salsa20 Janssen–Lange–Schwabe– Simon: 106 ops broken 256 144 Smetsers "TweetNaCl" 147.2 PRESENT 128 includes essentially the follow 156 Skinny 256 162.75 Piccolo 128 implementation of Salsa20: 202.5 128 AES AES 256 283.5

2

Operation counts are a poor model of hardware cost worse model of software cost

Bit operations per bit of plaintext (assuming precomputed subkeys), not entirely listed in Skinny paper:

key	ops/bit	cipher	Pick a cip
256	54	Salsa20/8	How fast i
256	78	Salsa20/12	
128	88	Simon: 60 ops broken	First step
128	100	NOEKEON	Write sim
128	117	Skinny	a a Rorna
256	126	Salsa20	e.g. Dems
256	144	Simon: 106 ops broken	Janssen-L
128	147.2	PRESENT	Smetsers
256	156	Skinny	includes es
128	162.75	Piccolo	implement
128	202.5	AES	•
256	283.5	AES	

Operation counts are a

2

poor model of hardware cost, worse model of software cost.

- her: e.g., Salsa20. s Salsa20 software?
- in analysis:
- ple software.
- stein-van Gastel-
- ange-Schwabe-
- "TweetNaCl"
- ssentially the following
- tation of Salsa20:

ations per bit of plaintext ng precomputed subkeys), rely listed in Skinny paper: 2

s/bit	cipher
ŀ	Salsa20/8
3	Salsa20/12
3	Simon: 60 ops broken
)	NOEKEON
7	Skinny
ō	Salsa20
ł	Simon: 106 ops broken
7.2	PRESENT
5	Skinny
2.75	Piccolo
2.5	AES
8.5	AES

Operation counts are a

poor model of hardware cost, worse model of software cost. Pick a cipher: e.g., Salsa20. How fast is Salsa20 software?

First step in analysis: Write simple software.

e.g. Bernstein-van Gastel-Janssen–Lange–Schwabe– Smetsers "TweetNaCl" includes essentially the following implementation of Salsa20:

int	cr	[y]	ot	. (
cons	st	u	3	;
{				
u3	32	W	[1	(
in	ıt	i	,j	
FC	)R (	(i	<b>,</b> 4	
	x	5	*i	-
	x	_1-	⊦i	-
	x	6-	⊦i	-
	x	1:	1+	•
}				
FC	)R (	(i	, 1	(

bit of plaintext puted subkeys), in Skinny paper: 2

er a20/8 a20/12 on: 60 ops broken EKEON ny

a20

on: 106 ops broken SENT

ny

olo

Operation counts are a poor model of hardware cost, worse model of software cost.

Pick a cipher: e.g., Salsa20. How fast is Salsa20 software?

First step in analysis: Write simple software.

e.g. Bernstein-van Gastel-Janssen-Lange-Schwabe-Smetsers "TweetNaCI" includes essentially the following implementation of Salsa20:

int crypto\_core\_salsa const u8 \*in, const u8 { u32 w[16],x[16],y[1 int i,j,m; FOR(i,4) { x[5\*i] = 1d32(c+4)x[1+i] = 1d32(k+4)x[6+i] = 1d32(in+i)x[11+i] = 1d32(k+i)}

FOR(i, 16) y[i] = x

2	3	
ntext	Operation counts are a	int cryp
keys),	poor model of hardware cost,	const u8
paper:	worse model of software cost.	{
	Pick a cipher: e.g. Salsa20	u32 w[
	How fast is Salsa20 software?	int i,
hroken	First step in analysis:	FOR(i,
DIORCII	Write simple software.	x[5*:
	e.g. Bernstein–van Gastel–	x[1+:
c brokon	Janssen–Lange–Schwabe–	x[6+:
S DIOKEII	Smetsers "TweetNaCI"	x[11·
	includes essentially the following	}
	implementation of Salsa20:	
		FOR(i,
		1

to\_core\_salsa20(u8 \*out,

\*in,const u8 \*k,const u

- 16],x[16],y[16],t[4]; j,m;
- 4) {
- i] = ld32(c+4\*i);
- i] = 1d32(k+4\*i);
- i] = ld32(in+4\*i);
- +i] = 1d32(k+16+4\*i);

16) y[i] = x[i];

Operation counts are a poor model of hardware cost, worse model of software cost.

Pick a cipher: e.g., Salsa20. How fast is Salsa20 software?

First step in analysis: Write simple software.

e.g. Bernstein-van Gastel-Janssen–Lange–Schwabe– Smetsers "TweetNaCl" includes essentially the following implementation of Salsa20:

```
int crypto_core_salsa20(u8 *out,
const u8 *in,const u8 *k,const u8 *c)
{
  u32 w[16],x[16],y[16],t[4];
  int i,j,m;
  FOR(i,4) {
    x[5*i] = 1d32(c+4*i);
    x[1+i] = 1d32(k+4*i);
    x[6+i] = 1d32(in+4*i);
    x[11+i] = 1d32(k+16+4*i);
  }
  FOR(i, 16) y[i] = x[i];
```

on counts are a del of hardware cost, odel of software cost. 3

ipher: e.g., Salsa20. t is Salsa20 software?

p in analysis: mple software.

nstein-van Gastel-

-Lange–Schwabe–

s "TweetNaCl"

essentially the following ntation of Salsa20:

```
int crypto_core_salsa20(u8 *out,
const u8 *in,const u8 *k,const u8 *c)
{
  u32 w[16],x[16],y[16],t[4];
  int i,j,m;
 FOR(i,4) {
    x[5*i] = 1d32(c+4*i);
    x[1+i] = 1d32(k+4*i);
    x[6+i] = 1d32(in+4*i);
    x[11+i] = 1d32(k+16+4*i);
  }
 FOR(i, 16) y[i] = x[i];
```

FOR(i,2 FOR(j FOR t[1] t[2] t[3] t[0] FOR } FOR(m } FOR(i,1) return }

```
are a
dware cost,
ftware cost.
., Salsa20.
0 software?
SIS:
/are.
n Gastel-
hwabe-
JaCl"
/ the following
Salsa20:
```

```
int crypto_core_salsa20(u8 *out,
const u8 *in,const u8 *k,const u8 *c)
{
  u32 w[16],x[16],y[16],t[4];
  int i,j,m;
  FOR(i,4) {
    x[5*i] = 1d32(c+4*i);
    x[1+i] = 1d32(k+4*i);
    x[6+i] = ld32(in+4*i);
    x[11+i] = 1d32(k+16+4*i);
  }
  FOR(i,16) y[i] = x[i];
```

FOR(i,20) { FOR(j,4) { FOR(m,4) t[m] =t[1] ^= L32(t[( t[2] ^= L32(t[1 t[3] ^= L32(t[2 t[0] ^= L32(t[3 FOR(m,4) w[4\*j+ }  $FOR(m, 16) \times [m] =$ } FOR(i,16) st32(out return 0;

}

```
t.
```

?د

NING

```
4
                                                 FOR(i,20) {
int crypto_core_salsa20(u8 *out,
const u8 *in,const u8 *k,const u8 *c)
{
 u32 w[16],x[16],y[16],t[4];
  int i,j,m;
 FOR(i,4) {
    x[5*i] = 1d32(c+4*i);
                                                   }
    x[1+i] = 1d32(k+4*i);
    x[6+i] = 1d32(in+4*i);
    x[11+i] = 1d32(k+16+4*i);
                                                 }
 }
 FOR(i,16) y[i] = x[i];
                                                 return 0;
                                              }
```

FOR(j,4) {

FOR(m, 4) t[m] = x[(5\*j+4\*m)]

- t[1] = L32(t[0]+t[3], 7);
- t[2] ^= L32(t[1]+t[0], 9);
- t[3] ^= L32(t[2]+t[1],13);
- t[0] ^= L32(t[3]+t[2],18);
- FOR(m,4) w[4\*j+(j+m)%4] =
- FOR(m, 16) x[m] = w[m];

FOR(i,16) st32(out + 4 \* i,x[i

```
int crypto_core_salsa20(u8 *out,
const u8 *in,const u8 *k,const u8 *c)
{
 u32 w[16],x[16],y[16],t[4];
  int i,j,m;
  FOR(i,4) {
    x[5*i] = 1d32(c+4*i);
   x[1+i] = 1d32(k+4*i);
   x[6+i] = 1d32(in+4*i);
    x[11+i] = 1d32(k+16+4*i);
  }
```

}

FOR(i, 16) y[i] = x[i];

5 FOR(i,20) { FOR(j,4) { FOR(m,4) t[m] = x[(5\*j+4\*m)%16];t[1] ^= L32(t[0]+t[3], 7); t[2] ^= L32(t[1]+t[0], 9); t[3] ^= L32(t[2]+t[1],13); t[0] = L32(t[3]+t[2], 18);FOR(m,4) w[4\*j+(j+m)%4] = t[m];} FOR(m, 16) x[m] = w[m];} FOR(i,16) st32(out + 4 \* i,x[i] + y[i]); return 0;

```
core_salsa20(u8 *out,
*in,const u8 *k,const u8 *c)
6],x[16],y[16],t[4];
, m ;
) {
] = 1d32(c+4*i);
] = 1d32(k+4*i);
] = 1d32(in+4*i);
i] = 1d32(k+16+4*i);
```

6) y[i] = x[i];

FOR(i,20) { FOR(j,4) { FOR(m, 4) t[m] = x[(5\*j+4)]t[1] = L32(t[0]+t[3], 7]t[2] = L32(t[1]+t[0], 9t[3] ^= L32(t[2]+t[1],13 t[0] ^= L32(t[3]+t[2],18 FOR(m,4) w[4\*j+(j+m)%4]} FOR(m, 16) x[m] = w[m];} FOR(i,16) st32(out + 4 \* i,x return 0;

4

}

5	
	static co
	= "expand
*m)%16];	
);	int crypt
);	const u8
3);	{
3);	u8 z[16]
= t[m];	u32 u,i
	if (!b)
	FOR(i,1
	FOR(i,8
	while (
[i] + y[i]);	crypt

FOR(i

u = 1

20(u8 *out,
*k,const u8 *c)
6],t[4];
*i);
*i);
4*i);
16+4*i);
i];

}

FOR(i,20) { FOR(j,4) { FOR(m,4) t[m] = x[(5\*j+4\*m)%16];t[1] ^= L32(t[0]+t[3], 7); t[2] ^= L32(t[1]+t[0], 9); t[3] ^= L32(t[2]+t[1],13); t[0] ^= L32(t[3]+t[2],18); FOR(m,4) w[4\*j+(j+m)%4] = t[m];} FOR(m, 16) x[m] = w[m];} FOR(i,16) st32(out + 4 \* i,x[i] + y[i]); return 0;



4	5	
	FOR(i,20) {	static co
	FOR(j,4) {	= "expand
	FOR(m,4) t[m] = $x[(5*j+4*m)%16];$	
	t[1] ^= L32(t[0]+t[3], 7);	int crypt
	t[2] ^= L32(t[1]+t[0], 9);	const u8
	t[3] ^= L32(t[2]+t[1],13);	{
	t[0] ^= L32(t[3]+t[2],18);	u8 z[16
	FOR(m,4) w[4*j+(j+m)%4] = t[m];	u32 u,i
	}	if (!b)
	FOR(m, 16) $x[m] = w[m];$	FOR(i,1
	}	FOR(i,8
		while (
	FOR(i,16) st32(out + 4 * i,x[i] + y[i]);	crypt
	return 0;	FOR(i
	}	u = 1

8 \*c)

- onst u8 sigma[16]
- d 32-byte k";
- to\_stream\_salsa20\_xor(u8 \*m,u64 b,const u8 \*n,co
- 6],x[64];
- i;
- ) return 0;
- 16) z[i] = 0;
- 8) z[i] = n[i];
- (b >= 64) {
- to\_core\_salsa20(x,z,k,si
- i,64) c[i] = (m?m[i]:0)
- 1;

static const u8 sigma[16] = "expand 32-byte k"; int crypto\_stream\_salsa20\_xor(u8 \*c, const u8 \*m,u64 b,const u8 \*n,const u8 \*k) { u8 z[16],x[64]; u32 u,i; if (!b) return 0; FOR(i, 16) z[i] = 0;FOR(i,8) z[i] = n[i]; while (b >= 64) { crypto\_core\_salsa20(x,z,k,sigma); u = 1;

FOR(i,64) c[i] = (m?m[i]:0) ^ x[i];

5 ) { static const u8 sigma[16] ,4) { = "expand 32-byte k"; (m,4) t[m] = x[(5\*j+4\*m)%16];] = L32(t[0]+t[3], 7);int crypto\_stream\_salsa20\_xor(u ] = L32(t[1]+t[0], 9);const u8 \*m,u64 b,const u8 \*n,c ] ^= L32(t[2]+t[1],13); { ] = L32(t[3]+t[2],18);u8 z[16],x[64]; (m,4) w[4\*j+(j+m)%4] = t[m];u32 u,i; if (!b) return 0; ,16) x[m] = w[m];FOR(i, 16) z[i] = 0;FOR(i,8) z[i] = n[i]; while (b >= 64) { 6) st32(out + 4 \* i,x[i] + y[i]); crypto\_core\_salsa20(x,z,k,s FOR(i, 64) c[i] = (m?m[i]:0)); u = 1;

6	
	for (
	u +:
	z[i]
.8 *c,	u >:
onst u8 *k)	}
	b -= (
	C += (
	if (m)
	}
	if (b)
	crypto
	FOR(i
igma);	}
^ x[i];	return (
	}

```
 x[(5*j+4*m)%16]; 
]+t[3], 7);
]+t[0], 9);
2]+t[1],13);
3]+t[2],18);
-(j+m)%4] = t[m];
w[m];
+ 4 * i,x[i] + y[i]);
```

```
6
static const u8 sigma[16]
= "expand 32-byte k";
int crypto_stream_salsa20_xor(u8 *c,
const u8 *m,u64 b,const u8 *n,const u8 *k)
{
  u8 z[16],x[64];
  u32 u,i;
  if (!b) return 0;
  FOR(i, 16) z[i] = 0;
  FOR(i,8) z[i] = n[i];
  while (b >= 64) {
    crypto_core_salsa20(x,z,k,sigma);
    FOR(i,64) c[i] = (m?m[i]:0) ^ x[i];
    u = 1;
```

for (i = 8;i < 16 u += (u32) z[i] z[i] = u;u >>= 8; } b -= 64; c += 64; if (m) m += 64;} if (b) { crypto\_core\_salsa FOR(i,b) c[i] = 0} return 0;

}

```
5
                                                         6
             static const u8 sigma[16]
             = "expand 32-byte k";
)%16];
             int crypto_stream_salsa20_xor(u8 *c,
             const u8 *m,u64 b,const u8 *n,const u8 *k)
                                                                 }
             {
                                                                 b -= 64;
               u8 z[16],x[64];
                                                                 c += 64;
t[m];
               u32 u,i;
               if (!b) return 0;
                                                               }
               FOR(i, 16) z[i] = 0;
                                                               if (b) {
               FOR(i,8) z[i] = n[i];
               while (b >= 64) {
                                                               }
] + y[i]);
                 crypto_core_salsa20(x,z,k,sigma);
                 FOR(i,64) c[i] = (m?m[i]:0) ^ x[i];
                                                               return 0;
                 u = 1;
                                                            }
```

- for (i = 8;i < 16;++i) {</pre>
  - u += (u32) z[i];
  - z[i] = u;
  - u >>= 8;
- if (m) m += 64;

- crypto\_core\_salsa20(x,z,k,si
- FOR(i,b) c[i] = (m?m[i]:0) ^

```
6
                                                   for (i = 8;i < 16;++i) {</pre>
static const u8 sigma[16]
= "expand 32-byte k";
                                                     u += (u32) z[i];
                                                      z[i] = u;
int crypto_stream_salsa20_xor(u8 *c,
                                                     u >>= 8;
                                                   }
const u8 *m,u64 b,const u8 *n,const u8 *k)
{
                                                   b -= 64;
 u8 z[16],x[64];
                                                    c += 64;
                                                    if (m) m += 64;
 u32 u,i;
  if (!b) return 0;
                                                 }
                                                 if (b) {
 FOR(i, 16) z[i] = 0;
 FOR(i,8) z[i] = n[i];
  while (b >= 64) {
                                                 }
    crypto_core_salsa20(x,z,k,sigma);
   FOR(i,64) c[i] = (m?m[i]:0) ^ x[i];
                                                 return 0;
   u = 1;
                                               }
```

crypto\_core\_salsa20(x,z,k,sigma); FOR(i,b) c[i] = (m?m[i]:0) ^ x[i];

```
nst u8 sigma[16]
32-byte k";
p_stream_salsa20_xor(u8 *c,
*m,u64 b,const u8 *n,const u8 *k)
],x[64];
return 0;
6) z[i] = 0;
z[i] = n[i];
o >= 64) {
core_salsa20(x,z,k,sigma);
,64) c[i] = (m?m[i]:0) ^ x[i];
```

}

```
for (i = 8;i < 16;++i) {</pre>
    u += (u32) z[i];
    z[i] = u;
    u >>= 8;
  }
  b -= 64;
  c += 64;
  if (m) m += 64;
}
if (b) {
  crypto_core_salsa20(x,z,k,sigma);
  FOR(i,b) c[i] = (m?m[i]:0) ^ x[i];
}
return 0;
```

# Next ste For each compile and see

6	)
[16]	
.sa20_xor(u8 *c,	
ust u8 *n,const u8 *k)	
.];	
120(x,z,k,sigma);	
(m?m[i]:0) ^ x[i];	

```
for (i = 8;i < 16;++i) {</pre>
    u += (u32) z[i];
    z[i] = u;
    u >>= 8;
  }
  b -= 64;
  c += 64;
  if (m) m += 64;
}
if (b) {
  crypto_core_salsa20(x,z,k,sigma);
  FOR(i,b) c[i] = (m?m[i]:0) ^ x[i];
}
return 0;
```

# Next step in analy For each target CF compile the simple and see how fast i

```
6
                                                           7
                  for (i = 8;i < 16;++i) {</pre>
                    u += (u32) z[i];
                    z[i] = u;
                    u >>= 8;
*c,
nst u8 *k)
                  }
                  b -= 64;
                  c += 64;
                  if (m) m += 64;
                }
                if (b) {
                  crypto_core_salsa20(x,z,k,sigma);
                  FOR(i,b) c[i] = (m?m[i]:0) ^ x[i];
                }
gma);
^ x[i];
                return 0;
              }
```

# Next step in analysis: For each target CPU, compile the simple code,

and see how fast it is.

Next step in analysis: For each target CPU, compile the simple code, and see how fast it is.

Next step in analysis: For each target CPU, compile the simple code, and see how fast it is.

7

;

the analysis now ends.

# In compiler writer's fantasy world,

Next step in analysis: For each target CPU, compile the simple code, and see how fast it is. In compiler writer's fantasy world, the analysis now ends. "We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

7

;

i	=	8;i	<	16	;++i)	{
=	(1	132)	z	[i]	•	
]	=	u;				
>=	= 8	3;				
64	<b>!;</b>					
64	Ŀ;					
)	m	+= (	54;	, )		

```
core_salsa20(x,z,k,sigma);
,b) c[i] = (m?m[i]:0) ^ x[i];
```

);

Next step in analysis: For each target CPU, compile the simple code, and see how fast it is.

In compiler writer's fantasy world, the analysis now ends.

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

### Reality i

8

crypto_stream
salsa20 dolbeau/amd6
implementations
amd64 Skylake
amd64 HW+AES
amd64 IB+AES
amd64 Sandy Bridge
amd64 Piledriver
amd64 Bulldozer
amd64 C2 65nm
amd64 K10 32nm
amd64 K10 45nm
amd64 K10 65nm
amd64 Airmont
amd64 K8
amd64 Bobcat
amd64 Atom
x86 P4 Willamette
aarch64 Cortex-A57
aarch64 Cortex-A53
armeabi Cortex-A15
armeabi Cortex-A7
armeabi Cortex-A8
armeabi Cortex-A9+NE
armeabi Cortex-A9
armeabi Armada

Time

```
S;++i) {
```

;

7

```
a20(x,z,k,sigma);
(m?m[i]:0) ^ x[i];
```

Next step in analysis: For each target CPU, compile the simple code, and see how fast it is.

In compiler writer's fantasy world, the analysis now ends.

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

### Reality is more con

crypto_stream salsa20 dolbeau/and implementations	e/amd64-xmm6 md64_xmm6 164-xmm6int e/amd64-xm	e/amd64-: e/amd64-xmh armneon6 4-xmh2	e/merge prmneon3
amd64 Skylake			
amd64 HW+AES			
amd64 IB+AES			
amd64 Sandy Bridge			
amd64 Piledriver			
amd64 Bulldozer			
amd64 C2 65nm			
amd64 K10 32nm			
amd64 K10 45nm		<u></u>	$\rightarrow$
amd64 K10 65nm			$\leq 1$
amd64 Airmont			
amd64 K8			• • • • •
amd64 Bobcat			
amd64 Atom			
x86 P4 Willamette			
aarch64 Cortex-A57			
aarch64 Cortex-A53			
armeabi Cortex-A15			
armeabi Cortex-A7			
armeabi Cortex-A8			
armeabi Cortex-A9+N	EON		
armeabi Cortex-A9			
armeabi Armada			

Time

Next step in analysis: For each target CPU, compile the simple code, and see how fast it is.

In compiler writer's fantasy world, the analysis now ends.

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

8

crypto_stream salsa20 dolbeau/
implementations
amd64 Skylake
amd64 HW+AES
amd64 IB+AES
amd64 Sandy Bridge
amd64 Piledriver
amd64 Bulldozer
amd64 C2 65nm
amd64 K10 32nm
amd64 K10 45nm
amd64 K10 65nm
amd64 Airmont
amd64 K8
amd64 Bobcat
amd64 Atom
x86 P4 Willamette
aarch64 Cortex-A57
aarch64 Cortex-A53
armeabi Cortex-A15
armeabi Cortex-A7
armeabi Cortex-A8
armeabi Cortex-A9+
armeabi Cortex-A9
armeabi Armada

Time

gma); x[i];

### Reality is more complicated:



Next step in analysis: For each target CPU, compile the simple code, and see how fast it is.

In compiler writer's fantasy world, the analysis now ends.

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

# Reality is more complicated:

crypto_stream	e/amd64-xmm6 amd64_xmm6	e/amd64- e/amd64-xmm	-3
implementations	e/and04-xmmoint e/and04-xmm	-xmn2 5 e/a	ndo-2
amd64 Skylake	1	A Company	
amd64 HW+AES			-
amd64 IB+AES			- +
amd64 Sandy Bridge			
amd64 Piledriver			
amd64 Bulldozer			
amd64 C2 65nm			
amd64 K10 32nm			Ŧ
amd64 K10 45nm			-t
amd64 K10 65nm			$\overline{\langle}$
amd64 Airmont			
amd64 K8			
amd64 Bobcat			<b>[</b> -
amd64 Atom			1-
x86 P4 Willamette			1
aarch64 Cortex-A57			-ţ.
aarch64 Cortex-A53			1
armeabi Cortex-A15			
armeabi Cortex-A7			
armeabi Cortex-A8			•
armeabi Cortex-A9+N	IEON		
armeabi Cortex-A9			
armeabi Armada			
Time	400	96 8 <sup>.</sup>	192

8



9

### 16384

32768

p in analysis: target CPU, the simple code,

how fast it is.

iler writer's fantasy world, ysis now ends.

ne so close to optimal on chitectures that we can't n more without using NP e algorithms instead of s. We can only try to niggles here and there ne heuristics get wrong answers."

### Reality is more complicated:

8



# SUPER( includes of 563 c >20 imp Haswell: impleme gcc -03 is $6.15 \times$ Salsa20 merged

# with "m optimiza compiler

# e/x86-mmxhttps://bench.cr.yp.to 20161010

# sis:

- DU,
- e code,
- t is.
- s fantasy world, nds.

8

- e to optimal on that we can't hout using NP ns instead of only try to ere and there cs get
- wers."

# Reality is more complicated:

crypto_stream	e/amd64-xmm6	e/amd64-3 nd64-xm	x86_xmm5 e/regs e/amd64-	e/x86-pm l //e/x8	e/x	<sup>86-mmx</sup> https://bench.cr.y
salsa20 dolbeau/amd64 implementations	4-xmm6int armned e/amd64-xmm2 e/amd64-xmm5	n6 ermne ermd6-2	on d e/	ref e armneon2 ref	x86-athlon    e/x86-3    armneo	20161
amd64 Skylake						aylaku; 4 x 33106941; 2018 Mdel Core (5-8800; Jarabb; Elyikes (80940); eq
amd64 HW+AES					-[- <u> </u> - <u> </u> -	steph, 8 x 3000849; 2013 Intel Res ED-1276 V3; annels; HW+AEE (2063); av stick; 2 x 1700849; 2014 Intel Care H-2108, Annels; HW+AEE (2063); av
amd64 IB+AES		1+			<u>+</u>	
amd64 Sandy Bridge					- + - + - + -	Mannig, 2 x 2110/044, 2011 Intel Cure II: 2010/, arealist, Namiy Bridger (2042), and error; 2 x 3420/044, 2011 Intel Cure II: 2010, arealist, Namiy Bridger (2042), and error; 2 x 3420/044, 2011 Intel Cure II: 2010.
amd64 Piledriver				<b> </b>   -   -   -   -   -   -   -   -	-	Ministery 2 x 2020MH6, 2022 AMD A13-6468M, ansRi Printree (K1874), an
amd64 Bulldozer			-   <b>                             </b>	k	+-+	kylevil, 3 x 3000044; 2013 AMD A30-80395; anski, Printver (11955), rep kylevil, 4 x 31205944; 2011 AMD PX-8120; anski, Bulinker (802022), av
					+ - -	Lanue; 4 x 2381/M44; 2007 M44 Com 2 Quid Q4600; amiliti, C2 Blom (470); and
				━━┃	_    _   .	samata; 2 x 2.500010; 2000 term Law 2 tota femolo; avaita; C2 femolo((H1)) and stranks; 4 x 2420040; 2007 telef Care 2 Quid Q8000; avaita; C2 filters (MD); and
amd64 K10 32nm						kpinal; 4 $\times$ 2000/044; 2011 AMD A0-3000; amddd; K18 32em (300730); sep
amd64 K10 45nm		· Xt			11	ispinal) 6 x 38004043 2020 MMD Phenois 6 X6 10017; Jardell, K18 Abres (20152), sop Jardell 6 x 38004043 2020 MMD Phenois 6 X6 30017; Jardell K18 Abres (20152) au
						klassi 1 x 1700MH6 2010 AMD Almos II Neo K120, amatsi K18 dhuw (100900) wa
amd64 K10 65nm		$ \langle$				$g_{\rm T}(d) \approx 21800{\rm MeV}$ 2008 AVD Optimus 8084, and of, K18 05mm (10053), eq.
amd64 Airmont				▶ 1 1		part il a MODMER, 2018 1014 Calevos NELIO, availet, Anemari (10567), aq
				<#H	† -	
amd64 K8						$\max_{i} (2 \times 2000 MH) (2000 AMB Advice 65.32) (available HE (42002)) mag$
amd64 Bobcat						Minime, 2 x 34030449; 2013 AMD Or T06M, Amatel, Barbale (100133); seg Matkill; 2 x 14000449; 2013 AMD R-450, Amatel, Barbale (100133); seq Matkill; 2 x 14000449; 2013 AMD R-450, Amatel, Barbale (100133); seq
		· <del>\</del> - ·			╗┨╼┝╸╸╸╸	Merce; 1 × 1330/049 2211 Fd.et Alore 76536; 2ex864; Alore (10812); seg
amd64 Atom			111			Massas, 2 a 1868/MHG 2011 Milet Alare D2020, Arrield, Mare (18681), any Massas, 1 x 1000/MHG 2010 Milet Mass Public, America, Maser (10812), any
x86 P4 Willamette						$3\mathrm{aut}$ 1 $\times$ 2000.04%; 2001.1464 Pentury II, allit, P4.VII.Saveller (U2), au
aarch64 Cortex-A57						Simbardal, 6 x 2758949; 2018 IVAGA, Tepo X3; APCHG, Carlier AST (6189671); eq
aarch64 Cortex-A53						parts 4 × 2000/041/ 2018 divelops: 8808; sarchist; Curter-MR (41976334); say
armeabi Cortex-A15						Jahanakali, 8 v 2008/049, 2014 MATERA Tegra KI, aveniati, Curter-A20 (413/k070), eq
armeabi Cortex-A7						$\label{eq:poly} \mu \lambda ~ 4 \times 801096,~2015 Brandow BCMIRIA investor, Carlos & (11150710), eq.$
armeabi Cortex-A8				/		a, 1 = 1000446, 2013 TI Robert XAMIRHOUZCZ108, intendo, Carline-AB (4135082), reg
armeabi Cortex-A9-LNE			~~~~/-			
						-daar aand "aaaan" cumuuni kurgood) ad
armeabl Cortex-A9						klupe; 2 x 5000H6; 2010 WH68, Teps 204; investor, Carlando (10116090), eq
armeabi Armada						sanisti, 1 s 12028 Heij 2018 Mored Arrasis 305, arresis, Arrasis (M271311), sa
Time	4096	8192	16384	32768	65536	

9 SUPERCOP bencl includes 2064 imp p.to 010 of 563 cryptograp >20 implementati Haswell: Reasonal implementation co gcc -03 -fomitis  $6.15 \times$  slower th Salsa20 implement merged implement with "machine-ind

optimizations and compiler options:

### Reality is more complicated:

world,

al on can't g NP of to ere



SUPER includes of 563 >20 im Haswel implem gcc -0

9

merged with "n optimiz compile

SUPERCOP benchmarking to includes 2064 implementation of 563 cryptographic primitive >20 implementations of Sala

- Haswell: Reasonably simple
- implementation compiled wi
- gcc -O3 -fomit-frame-po
- is  $6.15 \times$  slower than fastest
- Salsa20 implementation.
- merged implementation
- with "machine-independent"
- optimizations and best of 12
- compiler options:  $4.52 \times$  slow

### Reality is more complicated:



SUPERCOP benchmarking toolkit includes 2064 implementations of 563 cryptographic primitives. >20 implementations of Salsa20.

9

10

Haswell: Reasonably simple ref implementation compiled with gcc -03 -fomit-frame-pointer is  $6.15 \times$  slower than fastest Salsa20 implementation.

merged implementation with "machine-independent" optimizations and best of 121 compiler options:  $4.52 \times$  slower.
### s more complicated:



9

SUPERCOP benchmarking toolkit includes 2064 implementations of 563 cryptographic primitives. >20 implementations of Salsa20. Haswell: Reasonably simple ref implementation compiled with gcc -O3 -fomit-frame-pointer

is  $6.15 \times$  slower than fastest Salsa20 implementation.

merged implementation with "machine-independent" optimizations and best of 121 compiler options:  $4.52 \times$  slower.

# Many m were dev to the ( impleme

# mplicated:



9

SUPERCOP benchmarking toolkit includes 2064 implementations of 563 cryptographic primitives. >20 implementations of Salsa20.

Haswell: Reasonably simple ref implementation compiled with gcc -O3 -fomit-frame-pointer is  $6.15 \times$  slower than fastest Salsa20 implementation.

merged implementation with "machine-independent" optimizations and best of 121 compiler options:  $4.52 \times$  slower.

# Many more impler were developed on to the (currently) implementation fo



SUPERCOP benchmarking toolkit includes 2064 implementations of 563 cryptographic primitives. >20 implementations of Salsa20. Haswell: Reasonably simple ref implementation compiled with gcc -O3 -fomit-frame-pointer is  $6.15 \times$  slower than fastest Salsa20 implementation.

merged implementation with "machine-independent" optimizations and best of 121 compiler options:  $4.52 \times$  slower. 10

# Many more implementations were developed on the way to the (currently) fastest implementation for this CPL

Haswell: Reasonably simple ref implementation compiled with gcc -O3 -fomit-frame-pointer is  $6.15 \times$  slower than fastest Salsa20 implementation.

merged implementation with "machine-independent" optimizations and best of 121 compiler options:  $4.52 \times$  slower. 10

Many more implementations were developed on the way to the (currently) fastest implementation for this CPU.

Haswell: Reasonably simple ref implementation compiled with gcc -O3 -fomit-frame-pointer is  $6.15 \times$  slower than fastest Salsa20 implementation.

merged implementation with "machine-independent" optimizations and best of 121 compiler options:  $4.52 \times$  slower. 10

Many more implementations were developed on the way to the (currently) fastest implementation for this CPU.

This is a common pattern. Very fast development cycle: modify the implementation, check that it still works, evaluate its performance.

Haswell: Reasonably simple ref implementation compiled with gcc -O3 -fomit-frame-pointer is  $6.15 \times$  slower than fastest Salsa20 implementation.

merged implementation with "machine-independent" optimizations and best of 121 compiler options:  $4.52 \times$  slower. 10

Many more implementations were developed on the way to the (currently) fastest implementation for this CPU.

This is a common pattern. Very fast development cycle: modify the implementation, check that it still works, evaluate its performance.

Results of each evaluation guide subsequent modifications.

Haswell: Reasonably simple ref implementation compiled with gcc -O3 -fomit-frame-pointer is  $6.15 \times$  slower than fastest Salsa20 implementation.

merged implementation with "machine-independent" optimizations and best of 121 compiler options:  $4.52 \times$  slower. 10

Many more implementations were developed on the way to the (currently) fastest implementation for this CPU.

This is a common pattern. Very fast development cycle: modify the implementation, check that it still works, evaluate its performance.

Results of each evaluation guide subsequent modifications.

The software engineer needs fast evaluation of performance.

COP benchmarking toolkit 2064 implementations ryptographic primitives. elementations of Salsa20.

10

Reasonably simple ref ntation compiled with

-fomit-frame-pointer slower than fastest implementation.

implementation achine-independent" itions and best of 121 options:  $4.52 \times$  slower. Many more implementations were developed on the way to the (currently) fastest implementation for this CPU.

This is a common pattern. Very fast development cycle: modify the implementation, check that it still works, evaluate its performance.

Results of each evaluation guide subsequent modifications.

The software engineer needs fast evaluation of performance.

### 11

# The unfo

Slow eva is often to this c nmarking toolkit lementations hic primitives. 10

ons of Salsa20.

oly simple ref

frame-pointer

an fastest

tation.

tation

ependent"

best of 121

 $4.52 \times$  slower.

Many more implementations were developed on the way to the (currently) fastest implementation for this CPU.

This is a common pattern. Very fast development cycle: modify the implementation, check that it still works, evaluate its performance.

Results of each evaluation guide subsequent modifications.

The software engineer needs fast evaluation of performance.

# The unfortunate realised of the second secon

toolkit ns ves. sa20. ref th inter

10

21 wer. Many more implementations were developed on the way to the (currently) fastest implementation for this CPU. This is a common pattern. Very fast development cycle: modify the implementation, check that it still works, evaluate its performance. Results of each evaluation guide subsequent modifications.

The software engineer needs fast evaluation of performance.

11

# The unfortunate reality:

# Slow evaluation of performa

# is often a huge obstacle

to this optimization process.

Many more implementations were developed on the way to the (currently) fastest implementation for this CPU.

This is a common pattern. Very fast development cycle: modify the implementation, check that it still works, evaluate its performance.

Results of each evaluation guide subsequent modifications.

The software engineer needs fast evaluation of performance. The unfortunate reality:

11

Slow evaluation of performance is often a huge obstacle to this optimization process.

Many more implementations were developed on the way to the (currently) fastest implementation for this CPU.

This is a common pattern. Very fast development cycle: modify the implementation, check that it still works, evaluate its performance.

Results of each evaluation guide subsequent modifications.

The software engineer needs fast evaluation of performance. The unfortunate reality:

11

Slow evaluation of performance is often a huge obstacle to this optimization process.

When performance evaluation is too slow, the software engineer has to switch context, and then switching back to optimization produces severe cache misses inside software engineer's brain. ("I'm out of the zone.")

Many more implementations were developed on the way to the (currently) fastest implementation for this CPU.

This is a common pattern. Very fast development cycle: modify the implementation, check that it still works, evaluate its performance.

Results of each evaluation guide subsequent modifications.

The software engineer needs fast evaluation of performance. The unfortunate reality:

11

Slow evaluation of performance is often a huge obstacle to this optimization process.

When performance evaluation is too slow, the software engineer has to switch context, and then switching back to optimization produces severe cache misses inside software engineer's brain. ("I'm out of the zone.") Often optimization is aborted.

("I'll try some other time.")

ore implementations veloped on the way currently) fastest ntation for this CPU.

common pattern. t development cycle: the implementation, at it still works,

its performance.

of each evaluation bsequent modifications.

tware engineer needs luation of performance. The unfortunate reality:

11

Slow evaluation of performance is often a huge obstacle to this optimization process.

When performance evaluation is too slow, the software engineer has to switch context, and then switching back to optimization produces severe cache misses inside software engineer's brain. ("I'm out of the zone.")

Often optimization is aborted. ("I'll try some other time.")

# Goal of Speed u by speed "Optimi help opt

# nentations

11

the way

fastest

r this CPU.

pattern.

nent cycle:

nentation,

works,

mance.

aluation modifications.

gineer needs f performance. The unfortunate reality:

Slow evaluation of performance is often a huge obstacle to this optimization process.

When performance evaluation is too slow, the software engineer has to switch context, and then switching back to optimization produces severe cache misses inside software engineer's brain. ("I'm out of the zone.")

Often optimization is aborted. ("I'll try some other time.")

# Goal of this talk: Speed up the optin by speeding up be "Optimize benchm

help optimize opti

11

The unfortunate reality:

Slow evaluation of performance is often a huge obstacle to this optimization process.

When performance evaluation is too slow, the software engineer has to switch context, and then switching back to optimization produces severe cache misses inside software engineer's brain. ("I'm out of the zone.")

ons.

J.

eds ance. Often optimization is aborted. ("I'll try some other time.") Goal o Speed

12

"Optimize benchmarking to help optimize optimize?"

Goal of this talk:

Speed up the optimization p

by speeding up benchmarkin

The unfortunate reality:

Slow evaluation of performance is often a huge obstacle to this optimization process.

When performance evaluation is too slow, the software engineer has to switch context, and then switching back to optimization produces severe cache misses inside software engineer's brain. ("I'm out of the zone.")

Often optimization is aborted. ("I'll try some other time.")

12

Goal of this talk: by speeding up benchmarking.

"Optimize benchmarking to help optimize optimization."

# Speed up the optimization process

The unfortunate reality:

Slow evaluation of performance is often a huge obstacle to this optimization process.

When performance evaluation is too slow, the software engineer has to switch context, and then switching back to optimization produces severe cache misses inside software engineer's brain. ("I'm out of the zone.")

Often optimization is aborted. ("I'll try some other time.")

12

Goal of this talk: by speeding up benchmarking.

"Optimize benchmarking to help optimize optimization."

What are the bottlenecks that really need speedups? Measure the benchmarking process to gain understanding.

"Benchmark benchmarking to help optimize benchmarking."

ortunate reality:

aluation of performance a huge obstacle ptimization process.

erformance evaluation is , the software engineer witch context, and then g back to optimization s severe cache misses oftware engineer's brain. it of the zone.")

ptimization is aborted. some other time.")

Goal of this talk:

12

Speed up the optimization process by speeding up benchmarking.

"Optimize benchmarking to help optimize optimization."

What are the bottlenecks that really need speedups? Measure the benchmarking process to gain understanding.

"Benchmark benchmarking to help optimize benchmarking."

# Accessin

# The soft on his la performa

eality:

- <sup>f</sup> performance stacle
- n process.
- e evaluation is vare engineer ext, and then optimization
- ache misses
- gineer's brain.
- one.")
- n is aborted.
- er time.")

Goal of this talk:

12

Speed up the optimization process by speeding up benchmarking.

"Optimize benchmarking to help optimize optimize?"

What are the bottlenecks that really need speedups? Measure the benchmarking process to gain understanding.

"Benchmark benchmarking to help optimize benchmarking."

# Accessing different

# The software engine on his laptop, but performance on m

nce

12

on is eer

hen

on

S ain.

d.

Goal of this talk: Speed up the optimization process by speeding up benchmarking.

"Optimize benchmarking to help optimize optimization."

What are the bottlenecks that really need speedups? Measure the benchmarking process to gain understanding.

"Benchmark benchmarking to help optimize benchmarking."

13

The software engineer writes on his laptop, but cares abo performance on many more

# Accessing different CPUs

Goal of this talk:

Speed up the optimization process by speeding up benchmarking.

"Optimize benchmarking to help optimize optimization."

What are the bottlenecks that really need speedups? Measure the benchmarking process to gain understanding.

"Benchmark benchmarking to help optimize benchmarking."

13

Accessing different CPUs

on his laptop, but cares about

# The software engineer writes code performance on many more CPUs.

Goal of this talk:

Speed up the optimization process by speeding up benchmarking.

"Optimize benchmarking to help optimize optimization."

What are the bottlenecks that really need speedups? Measure the benchmarking process to gain understanding.

"Benchmark benchmarking to help optimize benchmarking."

13

Accessing different CPUs

on his laptop, but cares about

Or at least *should* care!

Surprisingly common failure: A paper with "faster algorithms" actually has slower algorithms running on faster processors.

# The software engineer writes code

14

performance on many more CPUs.

Goal of this talk:

Speed up the optimization process by speeding up benchmarking.

"Optimize benchmarking to help optimize optimization."

What are the bottlenecks that really need speedups? Measure the benchmarking process to gain understanding.

"Benchmark benchmarking to help optimize benchmarking."

13

Accessing different CPUs

on his laptop, but cares about

Or at least *should* care!

Surprisingly common failure: A paper with "faster algorithms" actually has slower algorithms running on faster processors.

Systematic fix: Optimize each algorithm, new or old, for older and newer processors.

- The software engineer writes code
- performance on many more CPUs.

this talk:

p the optimization process ling up benchmarking.

ze benchmarking to imize optimization."

e the bottlenecks lly need speedups? the benchmarking to gain understanding.

nark benchmarking to imize benchmarking."

13

The software engineer writes code on his laptop, but cares about performance on many more CPUs.

Or at least *should* care!

Surprisingly common failure: A paper with "faster algorithms" actually has slower algorithms running on faster processors.

Systematic fix: Optimize each algorithm, new or old, for older and newer processors. 14

# For each Find a n copy coo (assumir collect n

mization process nchmarking.

narking to mization."

lenecks

eedups?

nmarking

derstanding.

hmarking to chmarking."

# Accessing different CPUs

The software engineer writes code on his laptop, but cares about performance on many more CPUs.

Or at least *should* care!

Surprisingly common failure: A paper with "faster algorithms" actually has slower algorithms running on faster processors.

Systematic fix: Optimize each algorithm, new or old, for older and newer processors.

# For each target CI Find a machine wi copy code to that (assuming it's on ' collect measureme

process

13

lg.

۱g.

to "

# Accessing different CPUs

The software engineer writes code on his laptop, but cares about performance on many more CPUs.

Or at least *should* care!

Surprisingly common failure: A paper with "faster algorithms" actually has slower algorithms running on faster processors.

Systematic fix: Optimize each algorithm, new or old, for older and newer processors. For each target CPU: Find a machine with that Cl copy code to that machine (assuming it's on the Interne collect measurements there.

# Accessing different CPUs

The software engineer writes code on his laptop, but cares about performance on many more CPUs.

Or at least *should* care!

Surprisingly common failure: A paper with "faster algorithms" actually has slower algorithms running on faster processors.

Systematic fix: Optimize each algorithm, new or old, for older and newer processors. 14

For each target CPU: Find a machine with that CPU, copy code to that machine (assuming it's on the Internet), collect measurements there.

# Accessing different CPUs

The software engineer writes code on his laptop, but cares about performance on many more CPUs.

Or at least *should* care!

Surprisingly common failure: A paper with "faster algorithms" actually has slower algorithms running on faster processors.

Systematic fix: Optimize each algorithm, new or old, for older and newer processors. 14

For each target CPU: Find a machine with that CPU, copy code to that machine (assuming it's on the Internet), collect measurements there.

But, for security reasons, most machines on the Internet disallow access by default, except access by the owner.

# Accessing different CPUs

The software engineer writes code on his laptop, but cares about performance on many more CPUs.

Or at least *should* care!

Surprisingly common failure: A paper with "faster algorithms" actually has slower algorithms running on faster processors.

Systematic fix: Optimize each algorithm, new or old, for older and newer processors. 14

For each target CPU: Find a machine with that CPU, copy code to that machine (assuming it's on the Internet), collect measurements there.

But, for security reasons, most machines on the Internet disallow access by default, except access by the owner.

Solution #1: Each software engineer buys each CPU. This is expensive at high end, time-consuming at low end.

# g different CPUs

ware engineer writes code ptop, but cares about ance on many more CPUs. 14

ast *should* care!

igly common failure:

- with "faster algorithms"
- has slower algorithms
- on faster processors.

tic fix: Optimize

orithm, new or old,

and newer processors.

For each target CPU: Find a machine with that CPU, copy code to that machine (assuming it's on the Internet), collect measurements there.

But, for security reasons, most machines on the Internet disallow access by default, except access by the owner.

Solution #1: Each software engineer buys each CPU. This is expensive at high end, time-consuming at low end.

15

# Solution Poor cov

# t CPUs

neer writes code cares about any more CPUs. 14

care!

on failure: ter algorithms" r algorithms

processors.

otimize

ew or old,

er processors.

For each target CPU: Find a machine with that CPU, copy code to that machine (assuming it's on the Internet), collect measurements there.

But, for security reasons, most machines on the Internet disallow access by default, except access by the owner.

Solution #1: Each software engineer buys each CPU. This is expensive at high end, time-consuming at low end.

# Solution #2: Ama Poor coverage of (

s code ut CPUs. 14

nms" ٦S

)rs.

For each target CPU: Find a machine with that CPU, copy code to that machine (assuming it's on the Internet), collect measurements there. But, for security reasons, most machines on the Internet disallow access by default, except access by the owner.

Solution #1: Each software engineer buys each CPU. This is expensive at high end, time-consuming at low end.

15

# Solution #2: Amazon. Poor coverage of CPUs.

For each target CPU: Find a machine with that CPU, copy code to that machine (assuming it's on the Internet), collect measurements there.

But, for security reasons, most machines on the Internet disallow access by default, except access by the owner.

Solution #1: Each software engineer buys each CPU. This is expensive at high end, time-consuming at low end.

15

Solution #2: Amazon. Poor coverage of CPUs.

For each target CPU: Find a machine with that CPU, copy code to that machine (assuming it's on the Internet), collect measurements there.

But, for security reasons, most machines on the Internet disallow access by default, except access by the owner.

Solution #1: Each software engineer buys each CPU. This is expensive at high end, time-consuming at low end.

15

Solution #2: Amazon. Poor coverage of CPUs.

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

For each target CPU: Find a machine with that CPU, copy code to that machine (assuming it's on the Internet), collect measurements there.

But, for security reasons, most machines on the Internet disallow access by default, except access by the owner.

Solution #1: Each software engineer buys each CPU. This is expensive at high end, time-consuming at low end.

15

Solution #2: Amazon. Poor coverage of CPUs.

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

Solution #4: Figure out who has the right machines. (How?) Send email saying "Are you willing to run this code?" Slow; unreliable; scales badly.
target CPU: nachine with that CPU, de to that machine ng it's on the Internet), neasurements there.

security reasons, achines on the Internet access by default, ccess by the owner.

#1: Each software buys each CPU. expensive at high end, suming at low end.

Solution #2: Amazon. Poor coverage of CPUs.

15

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

Solution #4: Figure out who has the right machines. (How?) Send email saying "Are you willing to run this code?" Slow; unreliable; scales badly.

### 16

# Solution "Can I h Saves til

# DU:

th that CPU,

15

- machine
- the Internet),
- nts there.
- easons,
- the Internet
- default,
- he owner.
- n software
- n CPU.
- at high end,
- low end.

Solution #2: Amazon. Poor coverage of CPUs.

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

Solution #4: Figure out who has the right machines. (How?) Send email saying "Are you willing to run this code?" Slow; unreliable; scales badly. PU,

15

et),

net

d,

Solution #2: Amazon. Poor coverage of CPUs.

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

Solution #4: Figure out who has the right machines. (How?) Send email saying "Are you willing to run this code?" Slow; unreliable; scales badly.

16

# Solution #5: Send email say "Can I have an account?" Saves time but less reliable.

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

Solution #4: Figure out who has the right machines. (How?) Send email saying "Are you willing to run this code?" Slow; unreliable; scales badly.

16

Solution #5: Send email saying "Can I have an account?" Saves time but less reliable.

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

Solution #4: Figure out who has the right machines. (How?) Send email saying "Are you willing to run this code?" Slow; unreliable; scales badly.

16

Solution #5: Send email saying "Can I have an account?" Saves time but less reliable. Solution #6: eBACS. Good: One-time centralized

effort to find machines.

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

Solution #4: Figure out who has the right machines. (How?) Send email saying "Are you willing to run this code?" Slow; unreliable; scales badly.

16

Solution #5: Send email saying "Can I have an account?" Saves time but less reliable. Solution #6: eBACS. Good: One-time centralized effort to find machines. Good: For each code submission,

one-time centralized audit.

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

Solution #4: Figure out who has the right machines. (How?) Send email saying "Are you willing to run this code?" Slow; unreliable; scales badly.

Solution #5: Send email saying "Can I have an account?" Saves time but less reliable. Solution #6: eBACS. Good: One-time centralized effort to find machines. Good: For each code submission, one-time centralized audit. Good: High reliability, high coverage, built-in tests.

16

Solution #3: Compile farms, such as GCC Compile Farm. Coverage of CPUs is better but not good enough for crypto. Usual goals are OS coverage and architecture coverage.

Solution #4: Figure out who has the right machines. (How?) Send email saying "Are you willing to run this code?" Slow; unreliable; scales badly.

Solution #5: Send email saying "Can I have an account?" Saves time but less reliable. Solution #6: eBACS. Good: One-time centralized effort to find machines. Good: For each code submission, one-time centralized audit. Good: High reliability, high coverage, built-in tests. Bad: Much too slow.

16

#2: Amazon. verage of CPUs.

#3: Compile farms, GCC Compile Farm. e of CPUs is better good enough for crypto. bals are OS coverage itecture coverage.

#4: Figure out who right machines. (How?) nail saying "Are you o run this code?" reliable; scales badly.

Solution #5: Send email saying "Can I have an account?" Saves time but less reliable. Solution #6: eBACS. Good: One-time centralized

16

effort to find machines.

Good: For each code submission, one-time centralized audit.

Good: High reliability, high coverage, built-in tests.

Bad: Much too slow.

17

# The eBA

Software somethi

Software sends pa centraliz

eBACS integrate

eBACS I new SUI currently

azon.

CPUs.

npile farms,

16

pile Farm.

is better

igh for crypto.

5 coverage

overage.

re out who

nines. (How?)

"Are you

code?"

cales badly.

Solution #5: Send email saying "Can I have an account?" Saves time but less reliable.

Solution #6: eBACS.

Good: One-time centralized effort to find machines.

Good: For each code submission, one-time centralized audit.

Good: High reliability, high coverage, built-in tests.

Bad: Much too slow.

### The eBACS data f

Software engineer something to bend

Software engineer sends package by centralized account

eBACS manager a integrates into SU

integrates into 50

eBACS manager b new SUPERCOP currently 26-mega 16

pto.

0 w?)

**y.** 

Solution #5: Send email saying "Can I have an account?" Saves time but less reliable. Solution #6: eBACS.

Good: One-time centralized effort to find machines.

Good: For each code submission, one-time centralized audit.

Good: High reliability, high coverage, built-in tests.

Bad: Much too slow.

17

### The eBACS data flow

- Software engineer has impl: something to benchmark.
- Software engineer submits in sends package by email or (v centralized account) git pu
- eBACS manager audits impl integrates into SUPERCOP.
- eBACS manager builds new SUPERCOP package:
- currently 26-megabyte xz.

Solution #5: Send email saying "Can I have an account?" Saves time but less reliable.

Solution #6: eBACS.

Good: One-time centralized effort to find machines.

Good: For each code submission, one-time centralized audit.

Good: High reliability, high coverage, built-in tests.

Bad: Much too slow.

17

### The eBACS data flow

Software engineer has impl: something to benchmark.

Software engineer submits impl: sends package by email or (with centralized account) git push.

eBACS manager audits impl, integrates into SUPERCOP.

eBACS manager builds new SUPERCOP package: currently 26-megabyte xz.

#5: Send email saying ave an account?" me but less reliable.

#6: eBACS.

One-time centralized find machines.

For each code submission, e centralized audit.

ligh reliability, erage, built-in tests.

uch too slow.

# The eBACS data flow

17

Software engineer has impl: something to benchmark.

Software engineer submits impl: sends package by email or (with centralized account) git push.

eBACS manager audits impl, integrates into SUPERCOP.

eBACS manager builds new SUPERCOP package: currently 26-megabyte xz.

eBACS and ann Each ma waits un is suffici Each ma downloa **SUPER** stored o On a typ

millions

d email saying count?" 17

- s reliable.
- CS.
- entralized
- nines.
- ode submission, ed audit.
- ility,
- lt-in tests.
- SW.

# The eBACS data flow

Software engineer has impl: something to benchmark.

Software engineer submits impl: sends package by email or (with centralized account) git push.

eBACS manager audits impl, integrates into SUPERCOP.

eBACS manager builds new SUPERCOP package: currently 26-megabyte xz.

# eBACS manager used and announces particular

Each machine ope waits until the ma is sufficiently idle.

Each machine ope downloads SUPER

SUPERCOP scans stored on disk from On a typical highmillions of files, se 17

### The eBACS data flow

Software engineer has impl: something to benchmark.

Software engineer submits impl: sends package by email or (with centralized account) git push.

eBACS manager audits impl, integrates into SUPERCOP.

eBACS manager builds new SUPERCOP package: currently 26-megabyte xz.

18

Each machine operator waits until the machine is sufficiently idle.

Each machine operator downloads SUPERCOP, run

SUPERCOP scans data stored on disk from previous On a typical high-end CPU: millions of files, several GB.

ssion,

# eBACS manager uploads and announces package.

# The eBACS data flow

Software engineer has impl: something to benchmark.

Software engineer submits impl: sends package by email or (with centralized account) git push.

eBACS manager audits impl, integrates into SUPERCOP.

eBACS manager builds new SUPERCOP package: currently 26-megabyte xz. 18

eBACS manager uploads and announces package.

19

Each machine operator waits until the machine is sufficiently idle.

Each machine operator downloads SUPERCOP, runs it.

SUPERCOP scans data stored on disk from previous runs. On a typical high-end CPU: millions of files, several GB.

### ACS data flow

e engineer has impl: ng to benchmark.

e engineer submits impl: ickage by email or (with ed account) git push.

manager audits impl, es into SUPERCOP.

manager builds PERCOP package:

/ 26-megabyte xz.

eBACS manager uploads and announces package.

18

Each machine operator waits until the machine is sufficiently idle.

Each machine operator downloads SUPERCOP, runs it.

SUPERCOP scans data stored on disk from previous runs. On a typical high-end CPU: millions of files, several GB.

# For each **SUPER SUPER** working saves res Typically **SUPER** from thi 700-meg Machine data.gz

### flow

has impl: hmark.

submits impl: email or (with t) git push. 18

udits impl, PERCOP.

uilds

package:

byte xz.

eBACS manager uploads and announces package.

Each machine operator waits until the machine is sufficiently idle.

Each machine operator downloads SUPERCOP, runs it.

SUPERCOP scans data stored on disk from previous runs. On a typical high-end CPU: millions of files, several GB.

# For each new impl SUPERCOP comp SUPERCOP meas working compiled saves results on di Typically at least a SUPERCOP collect from this machine 700-megabyte dat Machine operator data.gz, annound

18

npl: with sh.

7

eBACS manager uploads and announces package.

Each machine operator waits until the machine is sufficiently idle.

Each machine operator downloads SUPERCOP, runs it.

SUPERCOP scans data stored on disk from previous runs. On a typical high-end CPU: millions of files, several GB.

19

For each new impl-compiler SUPERCOP compiles+tests

- SUPERCOP measures each working compiled impl, saves results on disk.
- Typically at least an hour.
- SUPERCOP collects all data
- from this machine, typically
- 700-megabyte data.gz.
- Machine operator uploads data.gz, announces it.

eBACS manager uploads and announces package.

Each machine operator waits until the machine is sufficiently idle.

Each machine operator downloads SUPERCOP, runs it.

SUPERCOP scans data stored on disk from previous runs. On a typical high-end CPU: millions of files, several GB.

19

For each new impl-compiler pair, SUPERCOP compiles+tests impl.

SUPERCOP measures each working compiled impl, saves results on disk.

Typically at least an hour.

SUPERCOP collects all data from this machine, typically 700-megabyte data.gz.

Machine operator uploads data.gz, announces it.

manager uploads ounces package.

achine operator til the machine ently idle.

achine operator ds SUPERCOP, runs it.

COP scans data n disk from previous runs. pical high-end CPU: of files, several GB.

For each new impl-compiler pair, SUPERCOP compiles+tests impl.

19

SUPERCOP measures each working compiled impl, saves results on disk.

Typically at least an hour.

SUPERCOP collects all data from this machine, typically 700-megabyte data.gz.

Machine operator uploads data.gz, announces it.

eBACS I data.gz

20

Databas

53% cur 47% arc

For each (or for c scripts p Typically

Web pag Under a

ploads ckage. 19

rator chine

rator RCOP, runs it.

data

n previous runs.

end CPU:

everal GB.

For each new impl-compiler pair, SUPERCOP compiles+tests impl. SUPERCOP measures each working compiled impl, saves results on disk. Typically at least an hour. SUPERCOP collects all data from this machine, typically 700-megabyte data.gz.

Machine operator uploads data.gz, announces it.

# eBACS manager c data.gz into cent

Database currently 53% current uncou 47% archives of su

For each new data (or for cross-cuttin scripts process all Typically an hour

Web pages are reg Under an hour.

	19	<sup>20</sup> For each new impl-compiler pair, SUPERCOP compiles+tests impl.	eBACS
		SUPERCOP measures each working compiled impl, saves results on disk.	Databa 53% cu 47% ar
s it.		Typically at least an hour. SUPERCOP collects all data from this machine, typically 700-megabyte data.gz. Machine operator uploads data.gz, announces it.	For eac (or for scripts Typical Web pa Under

### 6 manager copies gz into central databa

- ase currently uses 500
- urrent uncompressed of
- rchives of superseded
- ch new data.gz
- cross-cutting updates
- process all results.
- lly an hour per machi
- ages are regenerated. an hour.

For each new impl-compiler pair, SUPERCOP compiles+tests impl.

SUPERCOP measures each working compiled impl, saves results on disk.

Typically at least an hour.

SUPERCOP collects all data from this machine, typically 700-megabyte data.gz.

Machine operator uploads data.gz, announces it.

20

eBACS manager copies data.gz into central database.

Database currently uses 500GB: 53% current uncompressed data, 47% archives of superseded data.

For each new data.gz (or for cross-cutting updates): scripts process all results. Typically an hour per machine.

Web pages are regenerated. Under an hour.

new impl-compiler pair, COP compiles+tests impl. 20

COP measures each compiled impl, sults on disk.

/ at least an hour.

COP collects all data s machine, typically gabyte data.gz.

operator uploads z, announces it.

eBACS manager copies data.gz into central database.

Database currently uses 500GB: 53% current uncompressed data, 47% archives of superseded data.

For each new data.gz (or for cross-cutting updates): scripts process all results. Typically an hour per machine.

Web pages are regenerated. Under an hour.

21

In progre New dat All imple Some m measure "publish does for All com All chec All meas All table -compiler pair, oiles+tests impl. 20

ures each

impl,

sk.

an hour.

cts all data

, typically

a.gz.

uploads

ces it.

eBACS manager copies data.gz into central database.

Database currently uses 500GB: 53% current uncompressed data, 47% archives of superseded data.

For each new data.gz (or for cross-cutting updates): scripts process all results. Typically an hour per machine.

Web pages are regenerated. Under an hour.

### In progress: SUPE

### New database stor

### All impls ever sub

- Some metadata no
- measurements. Bi
- "publish results" f
- does force new me
- All compiled impls
- All checksums of a
- All measurements.
- All tables, graphs,

pair,	
impl.	

20

eBACS manager copies data.gz into central database.

Database currently uses 500GB: 53% current uncompressed data, 47% archives of superseded data.

For each new data.gz (or for cross-cutting updates): scripts process all results. Typically an hour per machine.

Web pages are regenerated. Under an hour.

21

### In progress: SUPERCOP 2

- New database stored central
- All impls ever submitted.
- Some metadata not affectin
- measurements. But turning
- "publish results" for an impl
- does force new measuremen
- All compiled impls.
- All checksums of outputs.
- All measurements.
- All tables, graphs, etc.

eBACS manager copies data.gz into central database.

Database currently uses 500GB: 53% current uncompressed data, 47% archives of superseded data.

For each new data.gz (or for cross-cutting updates): scripts process all results. Typically an hour per machine.

Web pages are regenerated. Under an hour.

In progress: SUPERCOP 2 New database stored centrally: All impls ever submitted. Some metadata not affecting measurements. But turning on "publish results" for an impl *does* force new measurements. All compiled impls. All checksums of outputs. All measurements.

21

All tables, graphs, etc.

manager copies z into central database.

e currently uses 500GB: rent uncompressed data, hives of superseded data.

new data.gz

ross-cutting updates):

rocess all results.

/ an hour per machine.

ges are regenerated. n hour.

# In progress: SUPERCOP 2

21

New database stored centrally:

All impls ever submitted.

Some metadata not affecting measurements. But turning on "publish results" for an impl *does* force new measurements.

All compiled impls.

All checksums of outputs.

All measurements.

All tables, graphs, etc.

22

When no Impl is p Each co to check Each wo pushed 1 (when tl Each me immedia If impl s Measure after cor opies ral database. 21

y uses 500GB: mpressed data, uperseded data.

a.gz

ng updates):

results.

per machine.

generated.

# In progress: SUPERCOP 2

New database stored centrally: All impls ever submitted.

Some metadata not affecting measurements. But turning on "publish results" for an impl *does* force new measurements.

All compiled impls.

All checksums of outputs.

All measurements.

All tables, graphs, etc.

When new impl is Impl is pushed to Each compiled im to checksum mach Each working com pushed to benchm (when they are sur Each measuremen immediately to sul If impl says "publi Measurements are after comparisons

ase.

21

GB:

data,

data.

5):

ne.

In progress: SUPERCOP 2

New database stored centrally: All impls ever submitted.

Some metadata not affecting measurements. But turning on "publish results" for an impl *does* force new measurements.

All compiled impls.

All checksums of outputs.

All measurements.

All tables, graphs, etc.

When new impl is submitted

22

Each compiled impl is pushe to checksum machines.

Each working compiled impl pushed to benchmark machi (when they are sufficiently id

Each measurement is availal immediately to submitter.

If impl says "publish results" Measurements are put online

after comparisons are done.

Impl is pushed to compile se

### In progress: SUPERCOP 2

New database stored centrally:

All impls ever submitted.

Some metadata not affecting measurements. But turning on "publish results" for an impl *does* force new measurements.

All compiled impls.

All checksums of outputs.

All measurements.

All tables, graphs, etc.

22

When new impl is submitted:

Impl is pushed to compile servers.

Each compiled impl is pushed to checksum machines.

Each working compiled impl is pushed to benchmark machines (when they are sufficiently idle).

Each measurement is available immediately to submitter.

If impl says "publish results": Measurements are put online after comparisons are done.

### ess: SUPERCOP 2

- abase stored centrally:
- s ever submitted.
- etadata not affecting ments. But turning on results" for an impl ce new measurements.
- piled impls.
- ksums of outputs.
- surements.
- s, graphs, etc.

When new impl is submitted:

22

Impl is pushed to compile servers.

Each compiled impl is pushed to checksum machines.

Each working compiled impl is pushed to benchmark machines (when they are sufficiently idle).

Each measurement is available immediately to submitter.

If impl says "publish results": Measurements are put online after comparisons are done.

### 23

# Wait, w

- No more there's r
- Critical
- Can a ro
- take ove Or corru
- from oth

# RCOP 2

red centrally:

22

mitted.

ot affecting ut turning on or an impl easurements.

, ) \_

outputs.

etc.

When new impl is submitted:

Impl is pushed to compile servers.

Each compiled impl is pushed to checksum machines.

Each working compiled impl is pushed to benchmark machines (when they are sufficiently idle).

Each measurement is available immediately to submitter.

If impl says "publish results": Measurements are put online after comparisons are done.

### Wait, what about

No more central a there's no time for

Critical integrity c Can a rogue code take over the mac Or corrupt benchn from other submit lly:

22

g on ts.

When new impl is submitted:

Impl is pushed to compile servers. Each compiled impl is pushed to checksum machines.

Each working compiled impl is pushed to benchmark machines (when they are sufficiently idle).

Each measurement is available immediately to submitter.

If impl says "publish results": Measurements are put online after comparisons are done.

23

### Wait, what about security?

- No more central auditing: there's no time for it.
- Critical integrity concerns:
- Can a rogue code submitter
- take over the machine?
- Or corrupt benchmarks
- from other submitters?

When new impl is submitted:

Impl is pushed to compile servers.

Each compiled impl is pushed to checksum machines.

Each working compiled impl is pushed to benchmark machines (when they are sufficiently idle).

Each measurement is available immediately to submitter.

If impl says "publish results": Measurements are put online after comparisons are done. 23

Wait, what about security?

24

No more central auditing: there's no time for it.

Critical integrity concerns: Can a rogue code submitter take over the machine? Or corrupt benchmarks from other submitters?
When new impl is submitted:

Impl is pushed to compile servers.

Each compiled impl is pushed to checksum machines.

Each working compiled impl is pushed to benchmark machines (when they are sufficiently idle).

Each measurement is available immediately to submitter.

If impl says "publish results": Measurements are put online after comparisons are done. 23

Wait, what about security?

24

No more central auditing: there's no time for it.

Critical integrity concerns: Can a rogue code submitter take over the machine? Or corrupt benchmarks from other submitters?

Concerns start before code is tested and measured: compilers have bugs, sometimes serious. When new impl is submitted:

Impl is pushed to compile servers.

Each compiled impl is pushed to checksum machines.

Each working compiled impl is pushed to benchmark machines (when they are sufficiently idle).

Each measurement is available immediately to submitter.

If impl says "publish results": Measurements are put online after comparisons are done. 23

Wait, what about security?

24

No more central auditing: there's no time for it.

Critical integrity concerns: Can a rogue code submitter take over the machine? Or corrupt benchmarks from other submitters?

Concerns start before code is tested and measured: compilers have bugs, sometimes serious.

Smaller availability concerns: e.g., Bitcoin mining.

- ew impl is submitted:
- oushed to compile servers.
- mpiled impl is pushed sum machines.
- orking compiled impl is to benchmark machines ney are sufficiently idle).
- easurement is available tely to submitter.
- ays "publish results": ments are put online mparisons are done.

### Wait, what about security?

23

No more central auditing: there's no time for it.

Critical integrity concerns: Can a rogue code submitter take over the machine? Or corrupt benchmarks from other submitters?

Concerns start before code is tested and measured: compilers have bugs, sometimes serious.

Smaller availability concerns: e.g., Bitcoin mining.

24

**SUPER OS-level** impl car cannot f **SUPER** pool of each cor machine Enforces for files in comp More dif integrity tables co

- submitted:
- compile servers.

23

- ol is pushed nines.
- piled impl is ark machines fficiently idle).
- t is available bmitter.
- sh results": put online are done.

## Wait, what about security?

No more central auditing: there's no time for it.

Critical integrity concerns: Can a rogue code submitter take over the machine? Or corrupt benchmarks from other submitters?

Concerns start before code is tested and measured: compilers have bugs, sometimes serious.

Smaller availability concerns: e.g., Bitcoin mining.

# SUPERCOP 1 set OS-level resource impl cannot open cannot fork any pr SUPERCOP 2 ma pool of uids and c each compile serve machine, benchma Enforces reasonab for files legitimate in compiling an im More difficult to e integrity policy for tables comparing i

]:

ervers.

23

ed

is nes dle).

ble

': e Wait, what about security?

No more central auditing: there's no time for it.

Critical integrity concerns: Can a rogue code submitter take over the machine? Or corrupt benchmarks from other submitters?

Concerns start before code is tested and measured: compilers have bugs, sometimes serious.

Smaller availability concerns: e.g., Bitcoin mining.

24

Enforce for files in com

More d ntegrit

## SUPERCOP 1 sets some OS-level resource limits:

- impl cannot open any files,
- cannot fork any processes.
- SUPERCOP 2 manages
- pool of uids and chroot jails
- each compile server, checksu
- machine, benchmark machir
- Enforces reasonable policy
- for files legitimately used
- in compiling an impl.
- More difficult to enforce:
- integrity policy for, e.g.,
- tables comparing impls.

## Wait, what about security?

No more central auditing: there's no time for it.

Critical integrity concerns: Can a rogue code submitter take over the machine? Or corrupt benchmarks from other submitters?

Concerns start before code is tested and measured: compilers have bugs, sometimes serious.

Smaller availability concerns: e.g., Bitcoin mining.

SUPERCOP 1 sets some OS-level resource limits: impl cannot open any files, cannot fork any processes. SUPERCOP 2 manages pool of uids and chroot jails on each compile server, checksum machine, benchmark machine. Enforces reasonable policy for files legitimately used in compiling an impl. More difficult to enforce: integrity policy for, e.g., tables comparing impls.

24