Picture credit: Rick Bowmer/AP

How to use
the new 65-megawatt
Bluffdale supercomputer:
a gentle introduction
to cryptanalysis

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimers

Picture credit: Rick Bowmer/AP

How to use
the new 65-megawatt
Bluffdale supercomputer:
a gentle introduction
to cryptanalysis

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimers

1. I don't work for NSA.

Picture credit: Rick Bowmer/AP

How to use
the new 65-megawatt
Bluffdale supercomputer:
a gentle introduction
to cryptanalysis

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimers

1. I don’t work for NSA.
2. NSA hasn’t told me anything.

Picture credit: Rick Bowmer/AP

How to use the new 65-megawatt Bluffdale supercomputer: a gentle introduction to cryptanalysis

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimers

1. I don't work for NSA.
2. NSA hasn't told me anything.
3. This is not a leak.

Picture credit: Rick Bowmer/AP

How to use
the new 65-megawatt
Bluffdale supercomputer:
a gentle introduction
to cryptanalysis

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimers

1. I don't work for NSA.
2. NSA hasn't told me anything.
3. This is not a leak.
4. I'm *assuming* that
NSA is not stupid.

Picture credit: Rick Bowmer/AP

How to use the new 65-megawatt Bluffdale supercomputer: a gentle introduction to cryptanalysis

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Disclaimers

1. I don't work for NSA.

2. NSA hasn't told me anything.

3. This is not a leak.

4. I'm *assuming* that NSA is not stupid.

5. Also *assuming* use of traditional transistors+wires, probably with some optics; plus long-term storage. Quantum computing would require different analysis.

edit: Rick Bowmer/AP

use
65-megawatt
e supercomputer:
introduction
analysis

ernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

Disclaimers

1. I don't work for NSA.

2. NSA hasn't told me anything.

3. This is not a leak.

4. I'm *assuming* that
NSA is not stupid.

5. Also *assuming* use of
traditional transistors+wires,
probably with some optics;
plus long-term storage.
Quantum computing would
require different analysis.

Cryptog

My miss
protect
against

owmer/AP

vatt

nputer:

ion

is at Chicago &

siteit Eindhoven

Disclaimers

1. I don't work for NSA.

2. NSA hasn't told me anything.

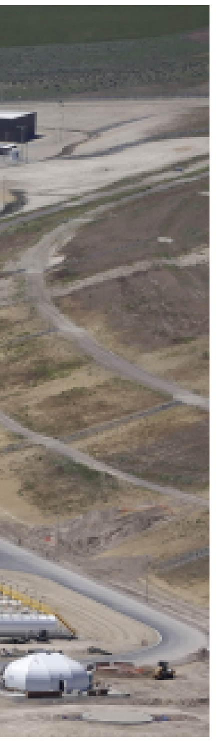
3. This is not a leak.

4. I'm *assuming* that
NSA is not stupid.

5. Also *assuming* use of
traditional transistors+wires,
probably with some optics;
plus long-term storage.
Quantum computing would
require different analysis.

Cryptographic cha

My mission: Crypt
protect every Inter
against espionage-



ago &
hoven

Disclaimers

1. I don't work for NSA.

2. NSA hasn't told me anything.

3. This is not a leak.

4. I'm *assuming* that NSA is not stupid.

5. Also *assuming* use of traditional transistors+wires, probably with some optics; plus long-term storage. Quantum computing would require different analysis.

Cryptographic challenges

My mission: Cryptographica
protect every Internet packe
against espionage+sabotage

Disclaimers

1. I don't work for NSA.

2. NSA hasn't told me anything.

3. This is not a leak.

4. I'm *assuming* that NSA is not stupid.

5. Also *assuming* use of traditional transistors+wires, probably with some optics; plus long-term storage. Quantum computing would require different analysis.

Cryptographic challenges

My mission: Cryptographically protect every Internet packet against espionage+sabotage.

Disclaimers

1. I don't work for NSA.

2. NSA hasn't told me anything.

3. This is not a leak.

4. I'm *assuming* that NSA is not stupid.

5. Also *assuming* use of traditional transistors+wires, probably with some optics; plus long-term storage. Quantum computing would require different analysis.

Cryptographic challenges

My mission: Cryptographically protect every Internet packet against espionage+sabotage.

User needs crypto to be fast on devices designed primarily for doing something else:

ers

't work for NSA.

hasn't told me anything.

is not a leak.

ssuming that
not stupid.

assuming use of
al transistors+wires,
with some optics;
g-term storage.
m computing would
different analysis.

Cryptographic challenges

My mission: Cryptographically protect every Internet packet against espionage+sabotage.

User needs crypto to be fast on devices designed primarily for doing something else:

User als
crypto t

Some ex

- 2009 e
  signat
  (small
- 2010 e
  signat
  (trivial
- 2012 e
  signat
  (some

r NSA.

d me anything.

ak.

hat

.

use of
ors+wires,
e optics;
rage.
ng would
nalysis.

Cryptographic challenges

My mission: Cryptographically protect every Internet packet against espionage+sabotage.

User needs crypto to be fast on devices designed primarily for doing something else:

User also needs crypto to be **secu**

Some examples of

- 2009 exploit of
  signatures in TI
  (small public co
- 2010 exploit of
  signatures in Pla
  (trivial—stupid
- 2012 exploit of
  signatures by Fla
  (somewhat large

hing.

,

Cryptographic challenges

My mission: Cryptographically protect every Internet packet against espionage+sabotage.

User needs crypto to be fast on devices designed primarily for doing something else:

User also needs crypto to be **secure**.

Some examples of crypto fai

- 2009 exploit of RSA-512 signatures in TI calculator (small public computation
- 2010 exploit of ECDSA signatures in PlayStation 3 (trivial—stupid Sony mista
- 2012 exploit of MD5-base signatures by Flame malwa (somewhat larger computa

Cryptographic challenges

My mission: Cryptographically protect every Internet packet against espionage+sabotage.

User needs crypto to be fast on devices designed primarily for doing something else:

User also needs crypto to be **secure**.

Some examples of crypto failing:

- 2009 exploit of RSA-512 signatures in TI calculators (small public computation);
- 2010 exploit of ECDSA signatures in PlayStation 3 (trivial—stupid Sony mistake);
- 2012 exploit of MD5-based signatures by Flame malware (somewhat larger computation).

Cryptographic challenges

My mission: Cryptographically protect every Internet packet against espionage+sabotage.

User needs crypto to be fast on devices designed primarily for doing something else:

User also needs crypto to be **secure**.

Some examples of crypto failing:

- 2009 exploit of RSA-512 signatures in TI calculators (small public computation);
- 2010 exploit of ECDSA signatures in PlayStation 3 (trivial—stupid Sony mistake);
- 2012 exploit of MD5-based signatures by Flame malware (somewhat larger computation).

Presumably many more examples not known to the public.

raphic challenges

ion: Cryptographically
every Internet packet
espionage+sabotage.

eds crypto to be fast
es designed primarily
g something else:

User also needs
crypto to be **secure**.

Some examples of crypto failing:

- 2009 exploit of RSA-512 signatures in TI calculators (small public computation);
- 2010 exploit of ECDSA signatures in PlayStation 3 (trivial—stupid Sony mistake);
- 2012 exploit of MD5-based signatures by Flame malware (somewhat larger computation).

Presumably many more examples not known to the public.

Critical c

Which c
fit the u
⇒ optim
cryptos
for each

llenges

tographically
rnet packet
+sabotage.

to be fast
ed primarily
ng else:

User also needs
crypto to be **secure**.

Some examples of crypto failing:

- 2009 exploit of RSA-512 signatures in TI calculators (small public computation);
- 2010 exploit of ECDSA signatures in PlayStation 3 (trivial—stupid Sony mistake);
- 2012 exploit of MD5-based signatures by Flame malware (somewhat larger computation).

Presumably many more examples not known to the public.

Critical questions:

Which cryptograp
fit the user's cost
⇒ optimize choic
cryptosystem +
for each user devi

lly
t
e.
t
y

User also needs
crypto to be **secure**.

Some examples of crypto failing:

- 2009 exploit of RSA-512 signatures in TI calculators (small public computation);
- 2010 exploit of ECDSA signatures in PlayStation 3 (trivial—stupid Sony mistake);
- 2012 exploit of MD5-based signatures by Flame malware (somewhat larger computation).

Presumably many more examples not known to the public.

Critical questions:

Which cryptographic system
fit the user's cost constraint
$\Rightarrow$ optimize choice of
cryptosystem + **algorithm**
for each user device.

User also needs crypto to be **secure**.

Some examples of crypto failing:

- 2009 exploit of RSA-512 signatures in TI calculators (small public computation);
- 2010 exploit of ECDSA signatures in PlayStation 3 (trivial—stupid Sony mistake);
- 2012 exploit of MD5-based signatures by Flame malware (somewhat larger computation).

Presumably many more examples not known to the public.

Critical questions:

Which cryptographic systems fit the user's cost constraints?
$\Rightarrow$ optimize choice of **cryptosystem** + **algorithm** for each user device.

User also needs
crypto to be **secure**.

Some examples of crypto failing:

- 2009 exploit of RSA-512 signatures in TI calculators (small public computation);
- 2010 exploit of ECDSA signatures in PlayStation 3 (trivial—stupid Sony mistake);
- 2012 exploit of MD5-based signatures by Flame malware (somewhat larger computation).

Presumably many more examples not known to the public.

Critical questions:

Which cryptographic systems fit the user's cost constraints?
$\Rightarrow$ optimize choice of **cryptosystem** + **algorithm** for each user device.

Which cryptographic systems can be broken by attackers?
$\Rightarrow$ optimize choice of **attack algorithm** + **device** for each cryptosystem.

User also needs
crypto to be **secure**.

Some examples of crypto failing:

- 2009 exploit of RSA-512 signatures in TI calculators (small public computation);
- 2010 exploit of ECDSA signatures in PlayStation 3 (trivial—stupid Sony mistake);
- 2012 exploit of MD5-based signatures by Flame malware (somewhat larger computation).

Presumably many more examples not known to the public.

Critical questions:

Which cryptographic systems fit the user's cost constraints?
$\Rightarrow$ optimize choice of **cryptosystem** + **algorithm** for each user device.

Which cryptographic systems can be broken by attackers?
$\Rightarrow$ optimize choice of **attack algorithm** + **device** for each cryptosystem.

Heavy interactions between high-level algorithms and low-level computer architecture.

o needs

o be **secure**.

xamples of crypto failing:

exploit of RSA-512

ures in TI calculators

public computation);

exploit of ECDSA

ures in PlayStation 3

l—stupid Sony mistake);

exploit of MD5-based

ures by Flame malware

what larger computation).

ably many more examples

wn to the public.

Critical questions:

Which cryptographic systems fit the user's cost constraints?
$\Rightarrow$ optimize choice of
cryptosystem + **algorithm**
for each user device.

Which cryptographic systems can be broken by attackers?
$\Rightarrow$ optimize choice of
attack algorithm + **device**
for each cryptosystem.

Heavy interactions between high-level algorithms and low-level computer architecture.

Theory v

Predictio

physicist

sometim

Commor

underlyi

calculati

re.

crypto failing:
RSA-512
calculators
mputation);
ECDSA
ayStation 3
Sony mistake);
MD5-based
ame malware
er computation).

more examples
public.

Critical questions:

Which cryptographic systems fit the user's cost constraints?
$\Rightarrow$ optimize choice of **cryptosystem** + **algorithm** for each user device.

Which cryptographic systems can be broken by attackers?
$\Rightarrow$ optimize choice of **attack algorithm** + **device** for each cryptosystem.

Heavy interactions between high-level algorithms and low-level computer architecture.

Theory vs. experim

Predictions made
physicists are ofte
sometimes wrong.

Common sources
underlying models
calculations from

iling:

s

);

3

ake);

d

are

ation).

mples

Critical questions:

Which cryptographic systems fit the user's cost constraints? $\Rightarrow$ optimize choice of **cryptosystem** + **algorithm** for each user device.

Which cryptographic systems can be broken by attackers? $\Rightarrow$ optimize choice of **attack algorithm** + **device** for each cryptosystem.

Heavy interactions between high-level algorithms and low-level computer architecture.

Theory vs. experiment

Predictions made by theoret
physicists are often disputed
sometimes wrong.

Common sources of error:
underlying models of physics
calculations from those mod

Critical questions:

Which cryptographic systems fit the user's cost constraints?
$\Rightarrow$ optimize choice of
cryptosystem + **algorithm**
for each user device.

Which cryptographic systems can be broken by attackers?
$\Rightarrow$ optimize choice of
attack algorithm + **device**
for each cryptosystem.

Heavy interactions between high-level algorithms and low-level computer architecture.

Theory vs. experiment

Predictions made by theoretical physicists are often disputed, sometimes wrong.

Common sources of error: underlying models of physics; calculations from those models.

Critical questions:

Which cryptographic systems fit the user's cost constraints?
$\Rightarrow$ optimize choice of
cryptosystem + **algorithm**
for each user device.

Which cryptographic systems can be broken by attackers?
$\Rightarrow$ optimize choice of
attack algorithm + **device**
for each cryptosystem.

Heavy interactions between high-level algorithms and low-level computer architecture.

Theory vs. experiment

Predictions made by theoretical physicists are often disputed, sometimes wrong.

Common sources of error: underlying models of physics; calculations from those models.

Experiments aren't perfect but catch many errors; resolve many disputes; provide raw data leading to new theories; build more confidence than theory alone can ever produce.

questions:

ryptographic systems
ser's cost constraints?
nize choice of
ystem + **algorithm**
user device.

ryptographic systems
roken by attackers?
nize choice of
algorithm + **device**
cryptosystem.

nteractions between
el algorithms and
l computer architecture.

Theory vs. experiment

Predictions made by theoretical physicists are often disputed, sometimes wrong.

Common sources of error: underlying models of physics; calculations from those models.

Experiments aren't perfect
but catch many errors;
resolve many disputes;
provide raw data
leading to new theories;
build more confidence than
theory alone can ever produce.

Is physic
Of cours

Every fie
theoretic
regardin
experime
measure
we comp

hic systems
constraints?
e of
algorithm
ce.

hic systems
attackers?
e of
$+$ **device**
tem.

s between
ms and
r architecture.

Theory vs. experiment

Predictions made by theoretical physicists are often disputed, sometimes wrong.

Common sources of error: underlying models of physics; calculations from those models.

Experiments aren't perfect
but catch many errors;
resolve many disputes;
provide raw data
leading to new theories;
build more confidence than
theory alone can ever produce.

Is physics uniquely
Of course not.

Every field of scien
theoreticians make
regarding observab
experimental scien
measure those phe
we compare the re

s
s?
s
e
ure.

Theory vs. experiment

Predictions made by theoretical
physicists are often disputed,
sometimes wrong.

Common sources of error:
underlying models of physics;
calculations from those models.

Experiments aren't perfect
but catch many errors;
resolve many disputes;
provide raw data
leading to new theories;
build more confidence than
theory alone can ever produce.

Is physics uniquely error-pro
Of course not.

Every field of science:
theoreticians make predictio
regarding observable phenon
experimental scientists
measure those phenomena;
we compare the results.

Theory vs. experiment

Predictions made by theoretical physicists are often disputed, sometimes wrong.

Common sources of error:
underlying models of physics;
calculations from those models.

Experiments aren't perfect
but catch many errors;
resolve many disputes;
provide raw data
leading to new theories;
build more confidence than
theory alone can ever produce.

Is physics uniquely error-prone?
Of course not.

Every field of science:
theoreticians make predictions
regarding observable phenomena;
experimental scientists
measure those phenomena;
we compare the results.

Theory vs. experiment

Predictions made by theoretical physicists are often disputed, sometimes wrong.

Common sources of error: underlying models of physics; calculations from those models.

Experiments aren't perfect but catch many errors; resolve many disputes; provide raw data leading to new theories; build more confidence than theory alone can ever produce.

Is physics uniquely error-prone? Of course not.

Every field of science: theoreticians make predictions regarding observable phenomena; experimental scientists measure those phenomena; we compare the results.

What if measurements are too expensive to carry out? Measurements start with scaled-down experiments, work up towards the scale of interest.

vs. experiment

ons made by theoretical
ts are often disputed,
nes wrong.

n sources of error:
ng models of physics;
ons from those models.

ents aren't perfect
h many errors;
many disputes;
raw data
to new theories;
ore confidence than
lone can ever produce.

Is physics uniquely error-prone?
Of course not.

Every field of science:
theoreticians make predictions
regarding observable phenomena;
experimental scientists
measure those phenomena;
we compare the results.

What if measurements are
too expensive to carry out?
Measurements start with
scaled-down experiments,
work up towards
the scale of interest.

Algorith
error-pro

Theoreti
regardin
These pr
disputed

ment

by theoretical
n disputed,

of error:
of physics;
those models.

t perfect
rrors;
utes;

eories;
ence than
ever produce.

Is physics uniquely error-prone?
Of course not.

Every field of science:
theoreticians make predictions
regarding observable phenomena;
experimental scientists
measure those phenomena;
we compare the results.

What if measurements are
too expensive to carry out?
Measurements start with
scaled-down experiments,
work up towards
the scale of interest.

Algorithm analysis
error-prone field o

Theoreticians mak
regarding algorith
These predictions
disputed, sometim

ical

,

s;

lels.

ce.

Is physics uniquely error-prone?
Of course not.

Every field of science:
theoreticians make predictions
regarding observable phenomena;
experimental scientists
measure those phenomena;
we compare the results.

What if measurements are
too expensive to carry out?
Measurements start with
scaled-down experiments,
work up towards
the scale of interest.

Algorithm analysis is anothe
error-prone field of science.

Theoreticians make predictic
regarding algorithm perform
These predictions are often
disputed, sometimes wrong.

Is physics uniquely error-prone?
Of course not.

Every field of science:
theoreticians make predictions
regarding observable phenomena;
experimental scientists
measure those phenomena;
we compare the results.

What if measurements are
too expensive to carry out?
Measurements start with
scaled-down experiments,
work up towards
the scale of interest.

Algorithm analysis is another
error-prone field of science.

Theoreticians make predictions
regarding algorithm performance.
These predictions are often
disputed, sometimes wrong.

Is physics uniquely error-prone?
Of course not.

Every field of science:
theoreticians make predictions
regarding observable phenomena;
experimental scientists
measure those phenomena;
we compare the results.

What if measurements are
too expensive to carry out?
Measurements start with
scaled-down experiments,
work up towards
the scale of interest.

Algorithm analysis is another
error-prone field of science.

Theoreticians make predictions
regarding algorithm performance.
These predictions are often
disputed, sometimes wrong.

Particularly error-prone:
cryptanalytic extrapolations
from an academic computation
to a serious real-world attack.

Is physics uniquely error-prone?
Of course not.

Every field of science:
theoreticians make predictions
regarding observable phenomena;
experimental scientists
measure those phenomena;
we compare the results.

What if measurements are
too expensive to carry out?
Measurements start with
scaled-down experiments,
work up towards
the scale of interest.

Algorithm analysis is another
error-prone field of science.

Theoreticians make predictions
regarding algorithm performance.
These predictions are often
disputed, sometimes wrong.

Particularly error-prone:
cryptanalytic extrapolations
from an academic computation
to a serious real-world attack.

We catch errors, resolve disputes
by carrying out experiments:
actually running these algorithms
on the largest scale we can.

cs uniquely error-prone?
se not.

eld of science:
cians make predictions
g observable phenomena;
ental scientists
those phenomena;
pare the results.

measurements are
ensive to carry out?
ements start with
own experiments,
towards
e of interest.

Algorithm analysis is another error-prone field of science.

Theoreticians make predictions regarding algorithm performance. These predictions are often disputed, sometimes wrong.

Particularly error-prone: cryptanalytic extrapolations from an academic computation to a serious real-world attack.

We catch errors, resolve disputes by carrying out experiments: actually running these algorithms on the largest scale we can.

1980s se
"QS" fa
costs 2^{10}

error-prone?

nce:
e predictions
ble phenomena;
ntists
enomena;
esults.

ments are
carry out?
rt with
iments,

st.

Algorithm analysis is another error-prone field of science.

Theoreticians make predictions regarding algorithm performance. These predictions are often disputed, sometimes wrong.

Particularly error-prone: cryptanalytic extrapolations from an academic computation to a serious real-world attack.

We catch errors, resolve disputes by carrying out experiments: actually running these algorithms on the largest scale we can.

1980s security eva
"QS" factorizatio
costs 2^{100} to brea

ne?

ns

nena;

Algorithm analysis is another error-prone field of science.

Theoreticians make predictions regarding algorithm performance. These predictions are often disputed, sometimes wrong.

Particularly error-prone: cryptanalytic extrapolations from an academic computation to a serious real-world attack.

We catch errors, resolve disputes by carrying out experiments: actually running these algorithms on the largest scale we can.

1980s security evaluation: “QS” factorization algorithm costs 2^{100} to break RSA-102

Algorithm analysis is another error-prone field of science.

Theoreticians make predictions regarding algorithm performance. These predictions are often disputed, sometimes wrong.

Particularly error-prone: cryptanalytic extrapolations from an academic computation to a serious real-world attack.

We catch errors, resolve disputes by carrying out experiments: actually running these algorithms on the largest scale we can.

1980s security evaluation: "QS" factorization algorithm costs 2^{100} to break RSA-1024.

Algorithm analysis is another error-prone field of science.

Theoreticians make predictions regarding algorithm performance. These predictions are often disputed, sometimes wrong.

Particularly error-prone: cryptanalytic extrapolations from an academic computation to a serious real-world attack.

We catch errors, resolve disputes by carrying out experiments: actually running these algorithms on the largest scale we can.

1980s security evaluation: "QS" factorization algorithm costs 2^{100} to break RSA-1024.

1990 Pollard: new "NFS".

Algorithm analysis is another error-prone field of science.

Theoreticians make predictions regarding algorithm performance. These predictions are often disputed, sometimes wrong.

Particularly error-prone: cryptanalytic extrapolations from an academic computation to a serious real-world attack.

We catch errors, resolve disputes by carrying out experiments: actually running these algorithms on the largest scale we can.

1980s security evaluation: "QS" factorization algorithm costs 2^{100} to break RSA-1024.

1990 Pollard: new "NFS".

1991 Adleman: NFS won't beat QS for RSA-1024.

Algorithm analysis is another error-prone field of science.

Theoreticians make predictions regarding algorithm performance. These predictions are often disputed, sometimes wrong.

Particularly error-prone: cryptanalytic extrapolations from an academic computation to a serious real-world attack.

We catch errors, resolve disputes by carrying out experiments: actually running these algorithms on the largest scale we can.

1980s security evaluation: "QS" factorization algorithm costs 2^{100} to break RSA-1024.

1990 Pollard: new "NFS".

1991 Adleman: NFS won't beat QS for RSA-1024.

Subsequent experiments $\Rightarrow$ NFS is much faster; maybe 2^{80}?

Algorithm analysis is another error-prone field of science.

Theoreticians make predictions regarding algorithm performance. These predictions are often disputed, sometimes wrong.

Particularly error-prone: cryptanalytic extrapolations from an academic computation to a serious real-world attack.

We catch errors, resolve disputes by carrying out experiments: actually running these algorithms on the largest scale we can.

1980s security evaluation: "QS" factorization algorithm costs 2^{100} to break RSA-1024.

1990 Pollard: new "NFS".

1991 Adleman: NFS won't beat QS for RSA-1024.

Subsequent experiments $\Rightarrow$ NFS is much faster; maybe 2^{80}?

Actual security of RSA-1024 is still a matter of dispute: e.g., 2009 Bos–Kaihara–Kleinjung–Lenstra–Montgomery oppose NIST's transition to RSA-2048.

m analysis is another
one field of science.

cians make predictions
g algorithm performance.
redictions are often
, sometimes wrong.

arly error-prone:
alytic extrapolations
academic computation
ous real-world attack.

h errors, resolve disputes
ing out experiments:
running these algorithms
argest scale we can.

1980s security evaluation:
"QS" factorization algorithm
costs 2^{100} to break RSA-1024.

1990 Pollard: new "NFS".

1991 Adleman: NFS
won't beat QS for RSA-1024.

Subsequent experiments ⇒
NFS is much faster; maybe 2^{80}?

Actual security of RSA-1024 is
still a matter of dispute: e.g.,
2009 Bos–Kaihara–Kleinjung–
Lenstra–Montgomery oppose
NIST's transition to RSA-2048.

The atta

Enough
should r
on amou
required

But can
this amo

s is another
f science.

ke predictions
m performance.
are often
es wrong.

prone:
apolations
computation
vorld attack.

esolve disputes
periments:
hese algorithms
e we can.

1980s security evaluation:
"QS" factorization algorithm
costs 2^{100} to break RSA-1024.

1990 Pollard: new "NFS".

1991 Adleman: NFS
won't beat QS for RSA-1024.

Subsequent experiments $\Rightarrow$
NFS is much faster; maybe 2^{80}?

Actual security of RSA-1024 is
still a matter of dispute: e.g.,
2009 Bos–Kaihara–Kleinjung–
Lenstra–Montgomery oppose
NIST's transition to RSA-2048.

<u>The attacker's sup</u>

Enough theory+ex
should reach conse
on amount of com
required to break a

But can the attack
this amount of cor

r

ons
ance.

ion
k.

utes
t
thms

1980s security evaluation:
"QS" factorization algorithm
costs 2^{100} to break RSA-1024.

1990 Pollard: new "NFS".

1991 Adleman: NFS
won't beat QS for RSA-1024.

Subsequent experiments $\Rightarrow$
NFS is much faster; maybe 2^{80}?

Actual security of RSA-1024 is
still a matter of dispute: e.g.,
2009 Bos–Kaihara–Kleinjung–
Lenstra–Montgomery oppose
NIST's transition to RSA-2048.

The attacker's supercomput

Enough theory+experiment
should reach consensus
on amount of computation
required to break a system.

But can the attacker perforr
this amount of computation

1980s security evaluation:
"QS" factorization algorithm
costs 2^{100} to break RSA-1024.

1990 Pollard: new "NFS".

1991 Adleman: NFS
won't beat QS for RSA-1024.

Subsequent experiments $\Rightarrow$
NFS is much faster; maybe 2^{80}?

Actual security of RSA-1024 is
still a matter of dispute: e.g.,
2009 Bos–Kaihara–Kleinjung–
Lenstra–Montgomery oppose
NIST's transition to RSA-2048.

The attacker's supercomputer

Enough theory+experiment
should reach consensus
on amount of computation
required to break a system.

But can the attacker perform
this amount of computation?

1980s security evaluation:
"QS" factorization algorithm
costs 2^{100} to break RSA-1024.

1990 Pollard: new "NFS".

1991 Adleman: NFS
won't beat QS for RSA-1024.

Subsequent experiments $\Rightarrow$
NFS is much faster; maybe 2^{80}?

Actual security of RSA-1024 is
still a matter of dispute: e.g.,
2009 Bos–Kaihara–Kleinjung–
Lenstra–Montgomery oppose
NIST's transition to RSA-2048.

The attacker's supercomputer

Enough theory+experiment
should reach consensus
on amount of computation
required to break a system.

But can the attacker perform
this amount of computation?

Hypothesize attacker resources.
This talk: $2 billion, 65MW.
Alternative: millions of
compromised Internet computers.

The interesting part: analyze
optimal use of those resources.

ecurity evaluation:
ctorization algorithm
^{00} to break RSA-1024.

llard: new "NFS".

lleman: NFS
eat QS for RSA-1024.

ent experiments ⇒
much faster; maybe 2^{80}?

ecurity of RSA-1024 is
atter of dispute: e.g.,
os–Kaihara–Kleinjung–
-Montgomery oppose
transition to RSA-2048.

The attacker's supercomputer

Enough theory+experiment should reach consensus on amount of computation required to break a system.

But can the attacker perform this amount of computation?

Hypothesize attacker resources.
This talk: \$2 billion, 65MW.
Alternative: millions of compromised Internet computers.

The interesting part: analyze optimal use of those resources.

Commun

Bill Dall
"Commu
more en

luation:
n algorithm
k RSA-1024.

"NFS".

FS
RSA-1024.

ments ⇒
er; maybe 2^{80}?

RSA-1024 is
spute: e.g.,
–Kleinjung–
ery oppose
to RSA-2048.

The attacker's supercomputer

Enough theory+experiment should reach consensus on amount of computation required to break a system.

But can the attacker perform this amount of computation?

Hypothesize attacker resources. This talk: \$2 billion, 65MW. Alternative: millions of compromised Internet computers.

The interesting part: analyze optimal use of those resources.

Communication vs

Bill Dally, 2013.06
"Communication t
more energy than

n
24.

4.

2^{80}?

4 is
g.,
g–
e
048.

The attacker's supercomputer

Enough theory+experiment should reach consensus on amount of computation required to break a system.

But can the attacker perform this amount of computation?

Hypothesize attacker resources. This talk: $2 billion, 65MW. Alternative: millions of compromised Internet computers.

The interesting part: analyze optimal use of those resources.

Communication vs. arithmet

Bill Dally, 2013.06.17: "Communication takes more energy than arithmetic

The attacker's supercomputer

Enough theory+experiment should reach consensus on amount of computation required to break a system.

But can the attacker perform this amount of computation?

Hypothesize attacker resources. This talk: \$2 billion, 65MW. Alternative: millions of compromised Internet computers.

The interesting part: analyze optimal use of those resources.

Communication vs. arithmetic

Bill Dally, 2013.06.17: "Communication takes more energy than arithmetic".

The attacker's supercomputer

Enough theory+experiment should reach consensus on amount of computation required to break a system.

But can the attacker perform this amount of computation?

Hypothesize attacker resources. This talk: $2 billion, 65MW. Alternative: millions of compromised Internet computers.

The interesting part: analyze optimal use of those resources.

Communication vs. arithmetic

Bill Dally, 2013.06.17: "Communication takes more energy than arithmetic".

Stephen S. Pawlowski, 2013.06.18: "The majority of energy that we spend today is on transferring data."

The attacker's supercomputer

Enough theory+experiment should reach consensus on amount of computation required to break a system.

But can the attacker perform this amount of computation?

Hypothesize attacker resources. This talk: \$2 billion, 65MW. Alternative: millions of compromised Internet computers.

The interesting part: analyze optimal use of those resources.

Communication vs. arithmetic

Bill Dally, 2013.06.17: "Communication takes more energy than arithmetic".

Stephen S. Pawlowski, 2013.06.18: "The majority of energy that we spend today is on transferring data."

Depends what you're doing!

Computations fundamentally vary in amount of communication (distance and volume) and amount of arithmetic.

acker's supercomputer

theory+experiment
each consensus
unt of computation
to break a system.

the attacker perform
ount of computation?

esize attacker resources.
k: $2 billion, 65MW.
ive: millions of
mised Internet computers.

eresting part: analyze
use of those resources.

Communication vs. arithmetic

Bill Dally, 2013.06.17:
"Communication takes
more energy than arithmetic".

Stephen S. Pawlowski,
2013.06.18: "The majority of
energy that we spend today
is on transferring data."

Depends what you're doing!

Computations fundamentally vary
in amount of communication
(distance and volume)
and amount of arithmetic.

Some al

Square r
n^2 arith

percomputer

xperiment
ensus
nputation
a system.

ker perform
mputation?

ker resources.
on, 65MW.
ns of
rnet computers.

rt: analyze
ose resources.

Communication vs. arithmetic

Bill Dally, 2013.06.17:
"Communication takes
more energy than arithmetic".

Stephen S. Pawlowski,
2013.06.18: "The majority of
energy that we spend today
is on transferring data."

Depends what you're doing!

Computations fundamentally vary
in amount of communication
(distance and volume)
and amount of arithmetic.

Some algorithms u

Square matrix-vec
n^2 arithmetic.

er

m

?

ces.

.

uters.

e

es.

Communication vs. arithmetic

Bill Dally, 2013.06.17:
"Communication takes
more energy than arithmetic".

Stephen S. Pawlowski,
2013.06.18: "The majority of
energy that we spend today
is on transferring data."

Depends what you're doing!

Computations fundamentally vary
in amount of communication
(distance and volume)
and amount of arithmetic.

Some algorithms using n^2 d

Square matrix-vector produc
n^2 arithmetic.

Communication vs. arithmetic

Bill Dally, 2013.06.17: "Communication takes more energy than arithmetic".

Stephen S. Pawlowski, 2013.06.18: "The majority of energy that we spend today is on transferring data."

Depends what you're doing!

Computations fundamentally vary in amount of communication (distance and volume) and amount of arithmetic.

Some algorithms using n^2 data:

Square matrix-vector product: n^2 arithmetic.

Communication vs. arithmetic

Bill Dally, 2013.06.17: "Communication takes more energy than arithmetic".

Stephen S. Pawlowski, 2013.06.18: "The majority of energy that we spend today is on transferring data."

Depends what you're doing!

Computations fundamentally vary in amount of communication (distance and volume) and amount of arithmetic.

Some algorithms using n^2 data:

Square matrix-vector product: n^2 arithmetic.

FFT for input size n^2: $n^2 \lg n$ arithmetic.

Communication vs. arithmetic

Bill Dally, 2013.06.17:
"Communication takes more energy than arithmetic".

Stephen S. Pawlowski, 2013.06.18: "The majority of energy that we spend today is on transferring data."

Depends what you're doing!

Computations fundamentally vary in amount of communication (distance and volume) and amount of arithmetic.

Some algorithms using n^2 data:

Square matrix-vector product: n^2 arithmetic.

FFT for input size n^2: $n^2 \lg n$ arithmetic.

Matrix-matrix product: typically n^3 arithmetic without Strassen etc.

Communication vs. arithmetic

Bill Dally, 2013.06.17: "Communication takes more energy than arithmetic".

Stephen S. Pawlowski, 2013.06.18: "The majority of energy that we spend today is on transferring data."

Depends what you're doing!

Computations fundamentally vary in amount of communication (distance and volume) and amount of arithmetic.

Some algorithms using n^2 data:

Square matrix-vector product: n^2 arithmetic.

FFT for input size n^2: $n^2 \lg n$ arithmetic.

Matrix-matrix product: typically n^3 arithmetic without Strassen etc.

Integrals in quantum chemistry, many common iterations, graph algorithms, etc.: n^4 arithmetic, sometimes more.

nication vs. arithmetic

y, 2013.06.17:
unication takes
ergy than arithmetic".

S. Pawlowski,
.18: "The majority of
hat we spend today
nsferring data."

s what you're doing!

ations fundamentally vary
nt of communication
e and volume)
ount of arithmetic.

Some algorithms using n^2 data:

Square matrix-vector product:
n^2 arithmetic.

FFT for input size n^2:
$n^2 \lg n$ arithmetic.

Matrix-matrix product:
typically n^3 arithmetic
without Strassen etc.

Integrals in quantum chemistry,
many common iterations,
graph algorithms, etc.:
n^4 arithmetic, sometimes more.

Chip are
is enoug
all data

s. arithmetic

.17:
takes
arithmetic".

wski,
majority of
end today
data."

're doing!

damentally vary
munication
me)
thmetic.

Some algorithms using n^2 data:

Square matrix-vector product:
n^2 arithmetic.

FFT for input size n^2:
$n^2 \lg n$ arithmetic.

Matrix-matrix product:
typically n^3 arithmetic
without Strassen etc.

Integrals in quantum chemistry,
many common iterations,
graph algorithms, etc.:
n^4 arithmetic, sometimes more.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2

tic

".

of

y vary

n

Some algorithms using n^2 data:

Square matrix-vector product:
n^2 arithmetic.

FFT for input size n^2:
$n^2 \lg n$ arithmetic.

Matrix-matrix product:
typically n^3 arithmetic
without Strassen etc.

Integrals in quantum chemistry,
many common iterations,
graph algorithms, etc.:
n^4 arithmetic, sometimes more.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2 FFT.

Some algorithms using n^2 data:

Square matrix-vector product:
n^2 arithmetic.

FFT for input size n^2:
$n^2 \lg n$ arithmetic.

Matrix-matrix product:
typically n^3 arithmetic
without Strassen etc.

Integrals in quantum chemistry,
many common iterations,
graph algorithms, etc.:
n^4 arithmetic, sometimes more.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2 FFT.

Some algorithms using n^2 data:

Square matrix-vector product:
n^2 arithmetic.

FFT for input size n^2:
$n^2 \lg n$ arithmetic.

Matrix-matrix product:
typically n^3 arithmetic
without Strassen etc.

Integrals in quantum chemistry,
many common iterations,
graph algorithms, etc.:
n^4 arithmetic, sometimes more.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2 FFT.

Chip area $n^{2+\epsilon}$
is also enough for
n^2 parallel ALUs.

Some algorithms using n^2 data:

Square matrix-vector product:
n^2 arithmetic.

FFT for input size n^2:
$n^2 \lg n$ arithmetic.

Matrix-matrix product:
typically n^3 arithmetic
without Strassen etc.

Integrals in quantum chemistry,
many common iterations,
graph algorithms, etc.:
n^4 arithmetic, sometimes more.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2 FFT.

Chip area $n^{2+\epsilon}$
is also enough for
n^2 parallel ALUs.

FFT takes time n^ϵ,
thanks to parallelism? No!
Routing the FFT data
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

Some algorithms using n^2 data:

Square matrix-vector product:
n^2 arithmetic.

FFT for input size n^2:
$n^2 \lg n$ arithmetic.

Matrix-matrix product:
typically n^3 arithmetic
without Strassen etc.

Integrals in quantum chemistry,
many common iterations,
graph algorithms, etc.:
n^4 arithmetic, sometimes more.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2 FFT.

Chip area $n^{2+\epsilon}$
is also enough for
n^2 parallel ALUs.

FFT takes time n^ϵ,
thanks to parallelism? No!
Routing the FFT data
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

1981 Brent–Kung: need $n^{1+\epsilon}$
even without wire delays.

gorithms using n^2 data:

matrix-vector product:
metic.

input size n^2:
arithmetic.

matrix product:
n^3 arithmetic
Strassen etc.

s in quantum chemistry,
ommon iterations,
gorithms, etc.:
metic, sometimes more.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2 FFT.

Chip area $n^{2+\epsilon}$
is also enough for
n^2 parallel ALUs.

FFT takes time n^{ϵ},
thanks to parallelism? No!
Routing the FFT data
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

1981 Brent–Kung: need $n^{1+\epsilon}$
even without wire delays.

Chip are
is enoug
several

Routing
occupies
for time

Typical
also occ
for time

Closer lo
the ALU
althoug

using n^2 data:

tor product:

n^2:

duct:
netic
etc.

um chemistry,
rations,
etc.:
netimes more.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2 FFT.

Chip area $n^{2+\epsilon}$
is also enough for
n^2 parallel ALUs.

FFT takes time n^ϵ,
thanks to parallelism? No!
Routing the FFT data
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

1981 Brent–Kung: need $n^{1+\epsilon}$
even without wire delays.

Chip area $n^{2+\epsilon}$
is enough to store
several $n \times n$ mat

Routing matrix pr
occupies area n^{2+}
for time $n^{1+\epsilon}$.

Typical n^3 arithme
also occupies n^2 A
for time $n^{1+\epsilon}$.

Closer look at ϵ:
the ALU cost dom
although not by m

ata:

ct:

stry,

ore.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2 FFT.

Chip area $n^{2+\epsilon}$
is also enough for
n^2 parallel ALUs.

FFT takes time n^ϵ,
thanks to parallelism? No!
Routing the FFT data
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

1981 Brent–Kung: need $n^{1+\epsilon}$
even without wire delays.

Chip area $n^{2+\epsilon}$
is enough to store
several $n \times n$ matrices.

Routing matrix product
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

Typical n^3 arithmetic
also occupies n^2 ALUs
for time $n^{1+\epsilon}$.

Closer look at ϵ:
the ALU cost dominates,
although not by much.

Chip area $n^{2+\epsilon}$
is enough to store
all data for size-n^2 FFT.

Chip area $n^{2+\epsilon}$
is also enough for
n^2 parallel ALUs.

FFT takes time n^ϵ,
thanks to parallelism? No!
Routing the FFT data
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

1981 Brent–Kung: need $n^{1+\epsilon}$
even without wire delays.

Chip area $n^{2+\epsilon}$
is enough to store
several $n \times n$ matrices.

Routing matrix product
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

Typical n^3 arithmetic
also occupies n^2 ALUs
for time $n^{1+\epsilon}$.

Closer look at ϵ:
the ALU cost dominates,
although not by much.

a $n^{2+\epsilon}$
h to store
for size-n^2 FFT.

a $n^{2+\epsilon}$
nough for
lel ALUs.

es time n^ϵ,
o parallelism? No!
the FFT data
area $n^{2+\epsilon}$
$n^{1+\epsilon}$.

ent–Kung: need $n^{1+\epsilon}$
hout wire delays.

Chip area $n^{2+\epsilon}$
is enough to store
several $n \times n$ matrices.

Routing matrix product
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

Typical n^3 arithmetic
also occupies n^2 ALUs
for time $n^{1+\epsilon}$.

Closer look at ϵ:
the ALU cost dominates,
although not by much.

$>$90% o
of typica
is spent
$<$10% o

Is Bluffd

^{2} FFT.

$^\epsilon$,

sm? No!

data

ϵ

need $n^{1+\epsilon}$

delays.

Chip area $n^{2+\epsilon}$
is enough to store
several $n \times n$ matrices.

Routing matrix product
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

Typical n^3 arithmetic
also occupies n^2 ALUs
for time $n^{1+\epsilon}$.

Closer look at ϵ:
the ALU cost dominates,
although not by much.

$>$90% of the cost
of typical supercor
is spent on commu
$<$10% on ALUs.

Is Bluffdale built t

Chip area $n^{2+\epsilon}$
is enough to store
several $n \times n$ matrices.

Routing matrix product
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

Typical n^3 arithmetic
also occupies n^2 ALUs
for time $n^{1+\epsilon}$.

Closer look at ϵ:
the ALU cost dominates,
although not by much.

>90% of the cost
of typical supercomputers
is spent on communication;
<10% on ALUs.

Is Bluffdale built this way?

Chip area $n^{2+\epsilon}$
is enough to store
several $n \times n$ matrices.

Routing matrix product
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

Typical n^3 arithmetic
also occupies n^2 ALUs
for time $n^{1+\epsilon}$.

Closer look at ϵ:
the ALU cost dominates,
although not by much.

$>$90% of the cost
of typical supercomputers
is spent on communication;
$<$10% on ALUs.

Is Bluffdale built this way?

Chip area $n^{2+\epsilon}$
is enough to store
several $n \times n$ matrices.

Routing matrix product
occupies area $n^{2+\epsilon}$
for time $n^{1+\epsilon}$.

Typical n^3 arithmetic
also occupies n^2 ALUs
for time $n^{1+\epsilon}$.

Closer look at ϵ:
the ALU cost dominates,
although not by much.

$>$90% of the cost
of typical supercomputers
is spent on communication;
$<$10% on ALUs.

Is Bluffdale built this way?
No; NSA is not stupid.

Doubling number of ALUs
would cost $<$10% extra.
Would $\approx$double performance
of matrix-matrix product
and heavier-arith computations.

NSA's computations have a mix
of heavy arith and heavy comm.

a $n^{2+\epsilon}$
h to store
$n \times n$ matrices.

matrix product
area $n^{2+\epsilon}$
$n^{1+\epsilon}$.

n^3 arithmetic
upies n^2 ALUs
$n^{1+\epsilon}$.

ok at ϵ:
cost dominates,
n not by much.

>90% of the cost
of typical supercomputers
is spent on communication;
<10% on ALUs.

Is Bluffdale built this way?
No; NSA is not stupid.

Doubling number of ALUs
would cost <10% extra.
Would ≈double performance
of matrix-matrix product
and heavier-arith computations.

NSA's computations have a mix
of heavy arith and heavy comm.

GPUs ha
but relat
commun
a few lo

Is Bluffd

rices.

oduct

ϵ

etic

ALUs

inates,

uch.

>90% of the cost
of typical supercomputers
is spent on communication;
<10% on ALUs.

Is Bluffdale built this way?
No; NSA is not stupid.

Doubling number of ALUs
would cost <10% extra.
Would ≈double performance
of matrix-matrix product
and heavier-arith computations.

NSA's computations have a mix
of heavy arith and heavy comm.

GPUs have many
but relatively little
communication ca
a few long wires t

Is Bluffdale built t

$>$90% of the cost
of typical supercomputers
is spent on communication;
$<$10% on ALUs.

Is Bluffdale built this way?
No; NSA is not stupid.

Doubling number of ALUs
would cost $<$10% extra.
Would $\approx$double performance
of matrix-matrix product
and heavier-arith computations.

NSA's computations have a mix
of heavy arith and heavy comm.

GPUs have many ALUs
but relatively little
communication capacity:
a few long wires to RAM.

Is Bluffdale built this way?

$>$90% of the cost
of typical supercomputers
is spent on communication;
$<$10% on ALUs.

Is Bluffdale built this way?
No; NSA is not stupid.

Doubling number of ALUs
would cost $<$10% extra.
Would $\approx$double performance
of matrix-matrix product
and heavier-arith computations.

NSA's computations have a mix
of heavy arith and heavy comm.

GPUs have many ALUs
but relatively little
communication capacity:
a few long wires to RAM.

Is Bluffdale built this way?

$>90\%$ of the cost
of typical supercomputers
is spent on communication;
$<10\%$ on ALUs.

Is Bluffdale built this way?
No; NSA is not stupid.

Doubling number of ALUs
would cost $<10\%$ extra.
Would $\approx$double performance
of matrix-matrix product
and heavier-arith computations.

NSA's computations have a mix
of heavy arith and heavy comm.

GPUs have many ALUs
but relatively little
communication capacity:
a few long wires to RAM.

Is Bluffdale built this way?
No; NSA is not stupid.

Adding communication
between adjacent ALUs
would cost very little.
Would drastically speed up
matrix-matrix product
and heavier-comm computations:
FFT, sorting, etc.

f the cost
al supercomputers
on communication;
n ALUs.

dale built this way?
A is not stupid.

g number of ALUs
ost <10% extra.
≈double performance
x-matrix product
vier-arith computations.

omputations have a mix
arith and heavy comm.

GPUs have many ALUs
but relatively little
communication capacity:
a few long wires to RAM.

Is Bluffdale built this way?
No; NSA is not stupid.

Adding communication
between adjacent ALUs
would cost very little.
Would drastically speed up
matrix-matrix product
and heavier-comm computations:
FFT, sorting, etc.

Docume
Intel Xe
and a fe
plus adj
commun

Is Bluffd

mputers
unication;

his way?
upid.

of ALUs
extra.
erformance
product
computations.

ons have a mix
heavy comm.

GPUs have many ALUs
but relatively little
communication capacity:
a few long wires to RAM.

Is Bluffdale built this way?
No; NSA is not stupid.

Adding communication
between adjacent ALUs
would cost very little.
Would drastically speed up
matrix-matrix product
and heavier-comm computations:
FFT, sorting, etc.

Documentation te
Intel Xeon Phi has
and a few long wi
plus adjacent one-
communication (ri

Is Bluffdale built t

e

ons.

mix
mm.

GPUs have many ALUs
but relatively little
communication capacity:
a few long wires to RAM.

Is Bluffdale built this way?
No; NSA is not stupid.

Adding communication
between adjacent ALUs
would cost very little.
Would drastically speed up
matrix-matrix product
and heavier-comm computations:
FFT, sorting, etc.

Documentation tells me tha
Intel Xeon Phi has many AL
and a few long wires to RAM
plus adjacent one-dimensio
communication (ring bus).

Is Bluffdale built this way?

GPUs have many ALUs
but relatively little
communication capacity:
a few long wires to RAM.

Is Bluffdale built this way?
No; NSA is not stupid.

Adding communication
between adjacent ALUs
would cost very little.
Would drastically speed up
matrix-matrix product
and heavier-comm computations:
FFT, sorting, etc.

Documentation tells me that
Intel Xeon Phi has many ALUs
and a few long wires to RAM
plus adjacent one-dimensional
communication (ring bus).

Is Bluffdale built this way?

GPUs have many ALUs
but relatively little
communication capacity:
a few long wires to RAM.

Is Bluffdale built this way?
No; NSA is not stupid.

Adding communication
between adjacent ALUs
would cost very little.
Would drastically speed up
matrix-matrix product
and heavier-comm computations:
FFT, sorting, etc.

Documentation tells me that
Intel Xeon Phi has many ALUs
and a few long wires to RAM
plus adjacent one-dimensional
communication (ring bus).

Is Bluffdale built this way?
No; NSA is not stupid.

Adding **two-dimensional grid**
would drastically speed up
heavy-comm computations.
e.g. 1977 Thompson–Kung.

Grid examples: MasPar; FPGAs.
But FPGAs have other problems.

ave many ALUs
tively little
nication capacity:
ng wires to RAM.

lale built this way?
A is not stupid.

communication
n adjacent ALUs
ost very little.
lrastically speed up
natrix product
vier-comm computations:
rting, etc.

Documentation tells me that Intel Xeon Phi has many ALUs and a few long wires to RAM **plus** adjacent one-dimensional communication (ring bus).

Is Bluffdale built this way?
No; NSA is not stupid.

Adding **two-dimensional grid** would drastically speed up heavy-comm computations.
e.g. 1977 Thompson–Kung.

Grid examples: MasPar; FPGAs.
But FPGAs have other problems.

Save eve
with 3D
e.g. 198

Huge en

2D allow
energy i
up to ve
3D is ha

Some lin
(most in
presuma
Progress
e.g., 4 ×
is often

ALUs
e
pacity:
o RAM.

his way?
upid.

ation
t ALUs
tle.
speed up
duct
computations:

Documentation tells me that Intel Xeon Phi has many ALUs and a few long wires to RAM **plus** adjacent one-dimensional communication (ring bus).

Is Bluffdale built this way? No; NSA is not stupid.

Adding **two-dimensional grid** would drastically speed up heavy-comm computations. e.g. 1977 Thompson–Kung.

Grid examples: MasPar; FPGAs. But FPGAs have other problems.

Save even more ti
with 3D arrangem
e.g. 1983 Rosenbe

Huge engineering

2D allows easy sca
energy input, heat
up to very large ch
3D is hard to scale

Some limited prog
(most interesting:
presumably used b
Progress often exa
e.g., $4 \times 16384 \times$
is often called "3D

tions:

Documentation tells me that Intel Xeon Phi has many ALUs and a few long wires to RAM **plus** adjacent one-dimensional communication (ring bus).

Is Bluffdale built this way? No; NSA is not stupid.

Adding **two-dimensional grid** would drastically speed up heavy-comm computations. e.g. 1977 Thompson–Kung.

Grid examples: MasPar; FPGAs. But FPGAs have other problems.

Save even more time with 3D arrangement of ALU e.g. 1983 Rosenberg.

Huge engineering challenge.

2D allows easy scaling of energy input, heat output up to very large chip area. 3D is hard to scale.

Some limited progress (most interesting: optics), presumably used by NSA. Progress often exaggerated: e.g., $4 \times 16384 \times 16384$ is often called “3D”.

Documentation tells me that Intel Xeon Phi has many ALUs and a few long wires to RAM **plus** adjacent one-dimensional communication (ring bus).

Is Bluffdale built this way? No; NSA is not stupid.

Adding **two-dimensional grid** would drastically speed up heavy-comm computations. e.g. 1977 Thompson–Kung.

Grid examples: MasPar; FPGAs. But FPGAs have other problems.

Save even more time with 3D arrangement of ALUs? e.g. 1983 Rosenberg.

Huge engineering challenge.

2D allows easy scaling of energy input, heat output up to very large chip area. 3D is hard to scale.

Some limited progress (most interesting: optics), presumably used by NSA. Progress often exaggerated: e.g., $4 \times 16384 \times 16384$ is often called "3D".

ntation tells me that
on Phi has many ALUs
w long wires to RAM
acent one-dimensional
ication (ring bus).

ale built this way?
A is not stupid.

two-dimensional grid
rastically speed up
omm computations.
77 Thompson–Kung.

mples: MasPar; FPGAs.
GAs have other problems.

Save even more time
with 3D arrangement of ALUs?
e.g. 1983 Rosenberg.

Huge engineering challenge.

2D allows easy scaling of
energy input, heat output
up to very large chip area.
3D is hard to scale.

Some limited progress
(most interesting: optics),
presumably used by NSA.
Progress often exaggerated:
e.g., $4 \times 16384 \times 16384$
is often called "3D".

Special v

Typical
between
better p
from AS
mass-ma

Some ex
ASICs b

lls me that
s many ALUs
res to RAM
-dimensional
ng bus).

his way?
upid.

nsional grid
speed up
putations.
son–Kung.

asPar; FPGAs.
other problems.

Save even more time with 3D arrangement of ALUs? e.g. 1983 Rosenberg.

Huge engineering challenge.

2D allows easy scaling of energy input, heat output up to very large chip area. 3D is hard to scale.

Some limited progress (most interesting: optics), presumably used by NSA. Progress often exaggerated: e.g., $4 \times 16384 \times 16384$ is often called “3D”.

<u>Special vs. general</u>

Typical cryptanaly
between 100× and
better performanc
from ASICs than f
mass-market CPU

Some exceptions,
ASICs bring massi

t
LUs
M
nal

rid

GAs.
lems.

Save even more time
with 3D arrangement of ALUs?
e.g. 1983 Rosenberg.

Huge engineering challenge.

2D allows easy scaling of
energy input, heat output
up to very large chip area.
3D is hard to scale.

Some limited progress
(most interesting: optics),
presumably used by NSA.
Progress often exaggerated:
e.g., $4 \times 16384 \times 16384$
is often called "3D".

<u>Special vs. general purpose</u>

Typical cryptanalytic arith:
between 100$\times$ and 1000$\times$
better performance per tran
from ASICs than from
mass-market CPUs, GPUs.

Some exceptions, but overal
ASICs bring massive speedu

Save even more time
with 3D arrangement of ALUs?
e.g. 1983 Rosenberg.

Huge engineering challenge.

2D allows easy scaling of
energy input, heat output
up to very large chip area.
3D is hard to scale.

Some limited progress
(most interesting: optics),
presumably used by NSA.
Progress often exaggerated:
e.g., $4 \times 16384 \times 16384$
is often called "3D".

<u>Special vs. general purpose</u>

Typical cryptanalytic arith:
between 100× and 1000×
better performance per transistor
from ASICs than from
mass-market CPUs, GPUs.

Some exceptions, but overall
ASICs bring massive speedup.

Save even more time
with 3D arrangement of ALUs?
e.g. 1983 Rosenberg.

Huge engineering challenge.

2D allows easy scaling of
energy input, heat output
up to very large chip area.
3D is hard to scale.

Some limited progress
(most interesting: optics),
presumably used by NSA.
Progress often exaggerated:
e.g., $4 \times 16384 \times 16384$
is often called "3D".

Special vs. general purpose

Typical cryptanalytic arith:
between $100\times$ and $1000\times$
better performance per transistor
from ASICs than from
mass-market CPUs, GPUs.

Some exceptions, but overall
ASICs bring massive speedup.

Only in cryptanalysis? No.
Estimated ASIC improvement
from preliminary scan of other
supercomputing arith problems:
usually $>10\times$, often $>100\times$.

en more time
arrangement of ALUs?
3 Rosenberg.

gineering challenge.

vs easy scaling of
nput, heat output
ry large chip area.
rd to scale.

mited progress
nteresting: optics),
bly used by NSA.
s often exaggerated:
: 16384×16384
called "3D".

Special vs. general purpose

Typical cryptanalytic arith: between $100\times$ and $1000\times$ better performance per transistor from ASICs than from mass-market CPUs, GPUs.

Some exceptions, but overall ASICs bring massive speedup.

Only in cryptanalysis? No. Estimated ASIC improvement from preliminary scan of other supercomputing arith problems: usually $>10\times$, often $>100\times$.

Frequent
chips sp
on deco

$\Rightarrow$ CPU
reduce i
by addin
apply sa
to multi

me
ent of ALUs?
erg.

challenge.

aling of
output
hip area.
e.

ress
optics),
y NSA.
ggerated:
16384
0".

Special vs. general purpose

Typical cryptanalytic arith: between 100× and 1000× better performance per transistor from ASICs than from mass-market CPUs, GPUs.

Some exceptions, but overall ASICs bring massive speedup.

Only in cryptanalysis? No. Estimated ASIC improvement from preliminary scan of other supercomputing arith problems: usually $>10\times$, often $>100\times$.

Frequent observat
chips spend area,
on decoding+sche

⇒ CPU/GPU desi
reduce insn-handli
by adding vectoriz
apply same instruc
to multiple data/t

Us?

Special vs. general purpose

Typical cryptanalytic arith: between 100× and 1000× better performance per transistor from ASICs than from mass-market CPUs, GPUs.

Some exceptions, but overall ASICs bring massive speedup.

Only in cryptanalysis? No. Estimated ASIC improvement from preliminary scan of other supercomputing arith problems: usually >10×, often >100×.

Frequent observation: chips spend area, time, ener on decoding+scheduling insn

⇒ CPU/GPU design trend: reduce insn-handling cost by adding vectorization— apply same instruction to multiple data/threads.

Special vs. general purpose

Typical cryptanalytic arith: between $100\times$ and $1000\times$ better performance per transistor from ASICs than from mass-market CPUs, GPUs.

Some exceptions, but overall ASICs bring massive speedup.

Only in cryptanalysis? No. Estimated ASIC improvement from preliminary scan of other supercomputing arith problems: usually $>10\times$, often $>100\times$.

Frequent observation: chips spend area, time, energy on decoding+scheduling insns.

$\Rightarrow$ CPU/GPU design trend: reduce insn-handling cost by adding vectorization— apply same instruction to multiple data/threads.

Special vs. general purpose

Typical cryptanalytic arith: between 100× and 1000× better performance per transistor from ASICs than from mass-market CPUs, GPUs.

Some exceptions, but overall ASICs bring massive speedup.

Only in cryptanalysis? No. Estimated ASIC improvement from preliminary scan of other supercomputing arith problems: usually $>10\times$, often $>100\times$.

Frequent observation: chips spend area, time, energy on decoding+scheduling insns.

$\Rightarrow$ CPU/GPU design trend: reduce insn-handling cost by adding vectorization— apply same instruction to multiple data/threads.

But this does nothing to reduce costs of reading data from reg file, writing data to reg file.

vs. general purpose

cryptanalytic arith:
100× and 1000×
erformance per transistor
ICs than from
arket CPUs, GPUs.

xceptions, but overall
ring massive speedup.

cryptanalysis? No.
ed ASIC improvement
eliminary scan of other
mputing arith problems:
>10×, often >100×.

Frequent observation:
chips spend area, time, energy
on decoding+scheduling insns.

⇒ CPU/GPU design trend:
reduce insn-handling cost
by adding vectorization—
apply same instruction
to multiple data/threads.

But this does nothing
to reduce costs of
reading data from reg file,
writing data to reg file.

Obvious
reduce t
combine
doing m
between

Example
to comp
CPU rea
compute

With se
CPU rea
writes; r
compute

l purpose

tic arith:
d 1000×
e per transistor
from
s, GPUs.

but overall
ve speedup.

ysis? No.
nprovement
can of other
rith problems:
en >100×.

Frequent observation:
chips spend area, time, energy
on decoding+scheduling insns.

⇒ CPU/GPU design trend:
reduce insn-handling cost
by adding vectorization—
apply same instruction
to multiple data/threads.

But this does nothing
to reduce costs of
reading data from reg file,
writing data to reg file.

Obvious strategy t
reduce these reg c
combine arith ope
doing more arith
between read and

Example: Build ci
to compute $xy +$
CPU reads regs x,
computes $xy + z$;

With separate mul
CPU reads x, y; c
writes; reads back
computes $xy + z$;

sistor

l
p.

nt
er
ms:
.

Frequent observation:
chips spend area, time, energy
on decoding+scheduling insns.

$\Rightarrow$ CPU/GPU design trend:
reduce insn-handling cost
by adding vectorization—
apply same instruction
to multiple data/threads.

But this does nothing
to reduce costs of
reading data from reg file,
writing data to reg file.

Obvious strategy to
reduce these reg costs:
combine arith operations,
doing more arith
between read and write.

Example: Build circuit
to compute $xy + z$.
CPU reads regs x, y, z;
computes $xy + z$; writes.

With separate mul, add:
CPU reads x, y; computes x
writes; reads back; reads z;
computes $xy + z$; writes.

Frequent observation:
chips spend area, time, energy
on decoding+scheduling insns.

$\Rightarrow$ CPU/GPU design trend:
reduce insn-handling cost
by adding vectorization—
apply same instruction
to multiple data/threads.

But this does nothing
to reduce costs of
reading data from reg file,
writing data to reg file.

Obvious strategy to
reduce these reg costs:
combine arith operations,
doing more arith
between read and write.

Example: Build circuit
to compute $xy + z$.
CPU reads regs x, y, z;
computes $xy + z$; writes.

With separate mul, add:
CPU reads x, y; computes xy;
writes; reads back; reads z;
computes $xy + z$; writes.

t observation:
end area, time, energy
ding+scheduling insns.

/GPU design trend:
nsn-handling cost
g vectorization—
me instruction
ple data/threads.

does nothing
e costs of
data from reg file,
data to reg file.

Obvious strategy to reduce these reg costs: combine arith operations, doing more arith between read and write.

Example: Build circuit to compute $xy+z$.
CPU reads regs x, y, z; computes $xy+z$; writes.

With separate mul, add:
CPU reads x, y; computes xy; writes; reads back; reads z; computes $xy+z$; writes.

Commor
evolved

Chip des
single-pr
eventual
circuit fo

ion:
time, energy
duling insns.

ign trend:
ng cost
ation—
ction
hreads.

ning

reg file,
g file.

Obvious strategy to reduce these reg costs: combine arith operations, doing more arith between read and write.

Example: Build circuit to compute $xy + z$. CPU reads regs x, y, z; computes $xy + z$; writes.

With separate mul, add: CPU reads x, y; computes xy; writes; reads back; reads z; computes $xy + z$; writes.

Common fp opera
evolved in this way

Chip designer saw
single-precision fp
eventually spent a
circuit for those m

gy
ns.

Obvious strategy to reduce these reg costs: combine arith operations, doing more arith between read and write.

Example: Build circuit to compute $xy + z$. CPU reads regs x, y, z; computes $xy + z$; writes.

With separate mul, add: CPU reads x, y; computes xy; writes; reads back; reads z; computes $xy + z$; writes.

Common fp operations evolved in this way.

Chip designer saw many single-precision fp muls, eventually spent area on circuit for those muls.

Obvious strategy to reduce these reg costs: combine arith operations, doing more arith between read and write.

Example: Build circuit to compute $xy + z$. CPU reads regs x, y, z; computes $xy + z$; writes.

With separate mul, add: CPU reads x, y; computes xy; writes; reads back; reads z; computes $xy + z$; writes.

Common fp operations evolved in this way.

Chip designer saw many single-precision fp muls, eventually spent area on circuit for those muls.

Obvious strategy to reduce these reg costs: combine arith operations, doing more arith between read and write.

Example: Build circuit to compute $xy + z$. CPU reads regs x, y, z; computes $xy + z$; writes.

With separate mul, add: CPU reads x, y; computes xy; writes; reads back; reads z; computes $xy + z$; writes.

Common fp operations evolved in this way.

Chip designer saw many single-precision fp muls, eventually spent area on circuit for those muls.

Then spent much more area to expand the multiplier to double-precision fp.

Obvious strategy to reduce these reg costs: combine arith operations, doing more arith between read and write.

Example: Build circuit to compute $xy + z$. CPU reads regs x, y, z; computes $xy + z$; writes.

With separate mul, add: CPU reads x, y; computes xy; writes; reads back; reads z; computes $xy + z$; writes.

Common fp operations evolved in this way.

Chip designer saw many single-precision fp muls, eventually spent area on circuit for those muls.

Then spent much more area to expand the multiplier to double-precision fp.

But people still run many single-precision computations. The multiplier transistors are **mostly sitting idle**.

strategy to
hese reg costs:
arith operations,
ore arith
read and write.

e: Build circuit
ute $xy + z$.
ds regs x, y, z;
es $xy + z$; writes.

arate mul, add:
ds x, y; computes xy;
eads back; reads z;
es $xy + z$; writes.

Common fp operations evolved in this way.

Chip designer saw many single-precision fp muls, eventually spent area on circuit for those muls.

Then spent much more area to expand the multiplier to double-precision fp.

But people still run many single-precision computations. The multiplier transistors are **mostly sitting idle**.

Another
Your app
mul-sub-
in its inn

Should C
include r
4 separa

to
osts:
rations,

write.

rcuit
z.
y, z;
writes.

l, add:
omputes xy;
; reads z;
writes.

Common fp operations
evolved in this way.

Chip designer saw many single-precision fp muls, eventually spent area on circuit for those muls.

Then spent much more area to expand the multiplier to double-precision fp.

But people still run many single-precision computations. The multiplier transistors are **mostly sitting idle**.

Another example:
Your application d
mul-sub-sub-sub-s
in its inner loop.

Should CPU desig
include mul circuit
4 separate sub circ

xy;

Common fp operations
evolved in this way.

Chip designer saw many
single-precision fp muls,
eventually spent area on
circuit for those muls.

Then spent much more area
to expand the multiplier
to double-precision fp.

But people still run many
single-precision computations.
The multiplier transistors are
mostly sitting idle.

Another example:
Your application does
mul-sub-sub-sub-sub
in its inner loop.

Should CPU designer
include mul circuit,
4 separate sub circuits?

Common fp operations
evolved in this way.

Chip designer saw many
single-precision fp muls,
eventually spent area on
circuit for those muls.

Then spent much more area
to expand the multiplier
to double-precision fp.

But people still run many
single-precision computations.
The multiplier transistors are
mostly sitting idle.

Another example:
Your application does
mul-sub-sub-sub-sub
in its inner loop.

Should CPU designer
include mul circuit,
4 separate sub circuits?

Common fp operations
evolved in this way.

Chip designer saw many
single-precision fp muls,
eventually spent area on
circuit for those muls.

Then spent much more area
to expand the multiplier
to double-precision fp.

But people still run many
single-precision computations.
The multiplier transistors are
mostly sitting idle.

Another example:
Your application does
mul-sub-sub-sub-sub
in its inner loop.

Should CPU designer
include mul circuit,
4 separate sub circuits?

Same CPU then runs
another application.
Subtraction circuits are
mostly sitting idle.

Common fp operations
evolved in this way.

Chip designer saw many
single-precision fp muls,
eventually spent area on
circuit for those muls.

Then spent much more area
to expand the multiplier
to double-precision fp.

But people still run many
single-precision computations.
The multiplier transistors are
mostly sitting idle.

Another example:
Your application does
mul-sub-sub-sub-sub
in its inner loop.

Should CPU designer
include mul circuit,
4 separate sub circuits?

Same CPU then runs
another application.
Subtraction circuits are
mostly sitting idle.

CPU designer says no,
reduces area per core.
$\Rightarrow$ Your application runs slowly.

n fp operations
in this way.

signer saw many
ecision fp muls,
ly spent area on
or those muls.

ent much more area
nd the multiplier
e-precision fp.

ple still run many
ecision computations.
ltiplier transistors are
sitting idle.

Another example:
Your application does
mul-sub-sub-sub-sub
in its inner loop.

Should CPU designer
include mul circuit,
4 separate sub circuits?

Same CPU then runs
another application.
Subtraction circuits are
mostly sitting idle.

CPU designer says no,
reduces area per core.
$\Rightarrow$ Your application runs slowly.

Many AS
beyond t

- Squari
  than n
- Skip m
- Reduc
  what i
- Add ve
  if appl
- etc.

tions
y.

many
muls,
rea on
uls.

more area
ltiplier
n fp.

n many
mputations.
nsistors are
e.

Another example:
Your application does
mul-sub-sub-sub-sub
in its inner loop.

Should CPU designer
include mul circuit,
4 separate sub circuits?

Same CPU then runs
another application.
Subtraction circuits are
mostly sitting idle.

CPU designer says no,
reduces area per core.
$\Rightarrow$ Your application runs slowly.

Many ASIC fp spe
beyond today's CP

- Squaring is chea
  than multiplicati
- Skip most norma
- Reduce precision
  what is actually
- Add very fast sq
  if application ne
- etc.

s.
e

Another example:
Your application does
mul-sub-sub-sub-sub
in its inner loop.

Should CPU designer
include mul circuit,
4 separate sub circuits?

Same CPU then runs
another application.
Subtraction circuits are
mostly sitting idle.

CPU designer says no,
reduces area per core.
$\Rightarrow$ Your application runs slowly.

Many ASIC fp speedups
beyond today's CPUs/GPUs

- Squaring is cheaper than multiplication.
- Skip most normalizations.
- Reduce precision to what is actually needed.
- Add very fast sqrt if application needs it.
- etc.

Another example:
Your application does
mul-sub-sub-sub-sub
in its inner loop.

Should CPU designer
include mul circuit,
4 separate sub circuits?

Same CPU then runs
another application.
Subtraction circuits are
mostly sitting idle.

CPU designer says no,
reduces area per core.
$\Rightarrow$ Your application runs slowly.

Many ASIC fp speedups
beyond today's CPUs/GPUs:

- Squaring is cheaper
  than multiplication.
- Skip most normalizations.
- Reduce precision to
  what is actually needed.
- Add very fast sqrt
  if application needs it.
- etc.

Another example:
Your application does
mul-sub-sub-sub-sub
in its inner loop.

Should CPU designer
include mul circuit,
4 separate sub circuits?

Same CPU then runs
another application.
Subtraction circuits are
mostly sitting idle.

CPU designer says no,
reduces area per core.
$\Rightarrow$ Your application runs slowly.

Many ASIC fp speedups
beyond today's CPUs/GPUs:

- Squaring is cheaper than multiplication.
- Skip most normalizations.
- Reduce precision to what is actually needed.
- Add very fast sqrt if application needs it.
- etc.

Cryptanalysis involves
many multiplications
but also a much wider
variety of operations.
Even larger ASIC speedups.

example:
plication does
-sub-sub-sub
ner loop.

CPU designer
mul circuit,
te sub circuits?

PU then runs
application.
tion circuits are
sitting idle.

signer says no,
area per core.
application runs slowly.

Many ASIC fp speedups
beyond today's CPUs/GPUs:

- Squaring is cheaper than multiplication.
- Skip most normalizations.
- Reduce precision to what is actually needed.
- Add very fast sqrt if application needs it.
- etc.

Cryptanalysis involves
many multiplications
but also a much wider
variety of operations.
Even larger ASIC speedups.

So NSA
for each

The sma
ASIC de
Not a se
for $2 bi

oes
ub

ner
t,
cuits?

uns
n.
ts are
e.

s no,
ore.
on runs slowly.

Many ASIC fp speedups
beyond today's CPUs/GPUs:

- Squaring is cheaper than multiplication.
- Skip most normalizations.
- Reduce precision to what is actually needed.
- Add very fast sqrt if application needs it.
- etc.

Cryptanalysis involves many multiplications but also a much wider variety of operations.
Even larger ASIC speedups.

So NSA builds AS
for each applicatio

The small problem
ASIC design effort
Not a serious issue
for $2 billion.

wly.

Many ASIC fp speedups beyond today's CPUs/GPUs:

- Squaring is cheaper than multiplication.
- Skip most normalizations.
- Reduce precision to what is actually needed.
- Add very fast sqrt if application needs it.
- etc.

Cryptanalysis involves many multiplications but also a much wider variety of operations. Even larger ASIC speedups.

So NSA builds ASICs for each application?

The small problem: ASIC design effort. Not a serious issue for $2 billion.

Many ASIC fp speedups
beyond today's CPUs/GPUs:

- Squaring is cheaper than multiplication.
- Skip most normalizations.
- Reduce precision to what is actually needed.
- Add very fast sqrt if application needs it.
- etc.

Cryptanalysis involves many multiplications but also a much wider variety of operations. Even larger ASIC speedups.

So NSA builds ASICs for each application?

The small problem: ASIC design effort. Not a serious issue for $2 billion.

Many ASIC fp speedups
beyond today's CPUs/GPUs:

- Squaring is cheaper than multiplication.
- Skip most normalizations.
- Reduce precision to what is actually needed.
- Add very fast sqrt if application needs it.
- etc.

Cryptanalysis involves many multiplications but also a much wider variety of operations. Even larger ASIC speedups.

So NSA builds ASICs for each application?

The small problem: ASIC design effort. Not a serious issue for $2 billion.

The big problem: Unpredictable application mix. NSA will want some agility to adapt to new computations and stop old computations.

Quantify using historical data: how long is an ASIC useful?

SIC fp speedups
today's CPUs/GPUs:
ng is cheaper
multiplication.
nost normalizations.
e precision to
s actually needed.
ery fast sqrt
ication needs it.

alysis involves
ultiplications
a much wider
of operations.
ger ASIC speedups.

So NSA builds ASICs
for each application?

The small problem:
ASIC design effort.
Not a serious issue
for $2 billion.

The big problem:
Unpredictable application mix.
NSA will want some agility
to adapt to new computations
and stop old computations.

Quantify using historical data:
how long is an ASIC useful?

Obvious
some AS
mix of **a**
integrate

Take a g
Add exa
XYZZY
plus som
Think ab
XYZZ? X
Still sim

New CP
Merge si
if not m

eedups
PUs/GPUs:
per
ion.
alizations.
n to
needed.
qrt
eds it.

lves
ons
wider
ns.
speedups.

So NSA builds ASICs
for each application?

The small problem:
ASIC design effort.
Not a serious issue
for $2 billion.

The big problem:
Unpredictable application mix.
NSA will want some agility
to adapt to new computations
and stop old computations.

Quantify using historical data:
how long is an ASIC useful?

Obvious solution f
some ASICs, plus
mix of **applicatio**
integrated circuits

Take a general-pu
Add exactly the bi
XYZZY needed by
plus some vectoriz
Think ahead, add
XYZZ? XZZY? XY
Still similar cost to

New CPU for each
Merge similar appl
if not much cost i

s:

So NSA builds ASICs
for each application?

The small problem:
ASIC design effort.
Not a serious issue
for $2 billion.

The big problem:
Unpredictable application mix.
NSA will want some agility
to adapt to new computations
and stop old computations.

Quantify using historical data:
how long is an ASIC useful?

Obvious solution for NSA:
some ASICs, plus heterogen
mix of **application-tuned**
integrated circuits (ATICs).

Take a general-purpose CPU
Add exactly the big insn
XYZZY needed by applicatio
plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each applicati
Merge similar applications
if not much cost in area.

So NSA builds ASICs
for each application?

The small problem:
ASIC design effort.
Not a serious issue
for $2 billion.

The big problem:
Unpredictable application mix.
NSA will want some agility
to adapt to new computations
and stop old computations.

Quantify using historical data:
how long is an ASIC useful?

Obvious solution for NSA:
some ASICs, plus heterogeneous
mix of **application-tuned**
integrated circuits (ATICs).

Take a general-purpose CPU.
Add exactly the big insn
XYZZY needed by application,
plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each application.
Merge similar applications
if not much cost in area.

builds ASICs
application?

all problem:
sign effort.
erious issue
illion.

problem:
ctable application mix.
l want some agility
t to new computations
o old computations.

y using historical data:
g is an ASIC useful?

Obvious solution for NSA:
some ASICs, plus heterogeneous
mix of **application-tuned**
integrated circuits (ATICs).

Take a general-purpose CPU.
Add exactly the big insn
XYZZY needed by application,
plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each application.
Merge similar applications
if not much cost in area.

1-slide B

Critical f
and imp

ICs
on?

n:

.

e

lication mix.
me agility
omputations
putations.

torical data:
IC useful?

Obvious solution for NSA:
some ASICs, plus heterogeneous
mix of **application-tuned**
integrated circuits (ATICs).

Take a general-purpose CPU.
Add exactly the big insn
XYZZY needed by application,
plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each application.
Merge similar applications
if not much cost in area.

1-slide Bluffdale us

Critical for algorit
and implementor:

ix.

ns

ta:

Obvious solution for NSA:
some ASICs, plus heterogeneous
mix of **application-tuned**
integrated circuits (ATICs).

Take a general-purpose CPU.
Add exactly the big insn
XYZZY needed by application,
plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each application.
Merge similar applications
if not much cost in area.

<u>1-slide Bluffdale user guide</u>

Critical for algorithm designe
and implementor:

Obvious solution for NSA:
some ASICs, plus heterogeneous mix of **application-tuned** integrated circuits (ATICs).

Take a general-purpose CPU.
Add exactly the big insn XYZZY needed by application, plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each application.
Merge similar applications *if* not much cost in area.

1-slide Bluffdale user guide

Critical for algorithm designer and implementor:

Obvious solution for NSA:
some ASICs, plus heterogeneous mix of **application-tuned** integrated circuits (ATICs).

Take a general-purpose CPU.
Add exactly the big insn XYZZY needed by application, plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each application.
Merge similar applications *if* not much cost in area.

1-slide Bluffdale user guide

Critical for algorithm designer and implementor:

Massive parallelism.

Obvious solution for NSA: some ASICs, plus heterogeneous mix of **application-tuned** integrated circuits (ATICs).

Take a general-purpose CPU. Add exactly the big insn XYZZY needed by application, plus some vectorization. Think ahead, add agility: XYZZ? XZZY? XYQZZY? Still similar cost to ASIC.

New CPU for each application. Merge similar applications *if* not much cost in area.

<u>1-slide Bluffdale user guide</u>

Critical for algorithm designer and implementor:

Massive parallelism.

Grid communication.

Obvious solution for NSA:
some ASICs, plus heterogeneous mix of **application-tuned** integrated circuits (ATICs).

Take a general-purpose CPU.
Add exactly the big insn XYZZY needed by application, plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each application.
Merge similar applications *if* not much cost in area.

1-slide Bluffdale user guide

Critical for algorithm designer and implementor:

Massive parallelism.

Grid communication.

Multiple instruction sets with very useful instructions.

Obvious solution for NSA:
some ASICs, plus heterogeneous mix of **application-tuned** integrated circuits (ATICs).

Take a general-purpose CPU.
Add exactly the big insn XYZZY needed by application, plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each application.
Merge similar applications *if* not much cost in area.

1-slide Bluffdale user guide

Critical for algorithm designer and implementor:

Massive parallelism.

Grid communication.

Multiple instruction sets with very useful instructions.

Some vectorization.

Obvious solution for NSA:
some ASICs, plus heterogeneous mix of **application-tuned** integrated circuits (ATICs).

Take a general-purpose CPU.
Add exactly the big insn XYZZY needed by application, plus some vectorization.
Think ahead, add agility:
XYZZ? XZZY? XYQZZY?
Still similar cost to ASIC.

New CPU for each application.
Merge similar applications *if* not much cost in area.

1-slide Bluffdale user guide

Critical for algorithm designer and implementor:

Massive parallelism.

Grid communication.

Multiple instruction sets with very useful instructions.

Some vectorization.

Occasional faults.

Obvious solution for NSA: some ASICs, plus heterogeneous mix of **application-tuned** integrated circuits (ATICs).

Take a general-purpose CPU. Add exactly the big insn XYZZY needed by application, plus some vectorization. Think ahead, add agility: XYZZ? XZZY? XYQZZY? Still similar cost to ASIC.

New CPU for each application. Merge similar applications *if* not much cost in area.

1-slide Bluffdale user guide

Critical for algorithm designer and implementor:

Massive parallelism.

Grid communication.

Multiple instruction sets with very useful instructions.

Some vectorization.

Occasional faults.

Need to understand cryptanalysis: ECM, sparse linear algebra, differentials, FFTs, much more.