The power of parallel computation

D. J. Bernstein

Thanks to:
University of Illinois at Chicago
NSF CCR–9983950
Alfred P. Sloan Foundation

How fast is sorting?

Input: array of n numbers.
Each number in $\{1, 2, \ldots, n^2\}$,
represented in binary.

Output: array of n numbers,
in increasing order,
represented in binary;
same multiset as input.

A machine is given the input
and computes the output.
How much time does it use?

on

is at Chicago
0
undation

How fast is sorting?

Input: array of n numbers.
Each number in $\{1, 2, \ldots, n^2\}$,
represented in binary.

Output: array of n numbers,
in increasing order,
represented in binary;
same multiset as input.

A machine is given the input
and computes the output.
How much time does it use?

The answer depen
how the machine

Possibility 1: The
"1-tape Turing ma
using selection sor

Specifically: The
a 1-dimensional ar
containing $\Theta(n)$ "
Each cell stores Θ

Input and output
stored in these cel

How fast is sorting?

Input: array of n numbers.
Each number in $\{1, 2, \ldots, n^2\}$,
represented in binary.

Output: array of n numbers,
in increasing order,
represented in binary;
same multiset as input.

A machine is given the input
and computes the output.
How much time does it use?

The answer depends on
how the machine works.

Possibility 1: The machine is a
"1-tape Turing machine
using selection sort."

Specifically: The machine has
a 1-dimensional array
containing $\Theta(n)$ "cells."
Each cell stores $\Theta(\lg n)$ bits.

Input and output are
stored in these cells.

g?

numbers.

$1, 2, \ldots, n^2\}$,

ary.

n numbers,

;

ary;

nput.

n the input

output.

oes it use?

The answer depends on how the machine works.

Possibility 1: The machine is a "1-tape Turing machine using selection sort."

Specifically: The machine has a 1-dimensional array containing $\Theta(n)$ "cells." Each cell stores $\Theta(\lg n)$ bits.

Input and output are stored in these cells.

The machine also

"head" moving thr

Head contains $\Theta($1

Head can see the

its current array p

perform arithmeti

move to adjacent

Selection sort: He

looks at each arra

picks up the large

moves it to the en

picks up the secon

etc.

The answer depends on
how the machine works.

Possibility 1: The machine is a
"1-tape Turing machine
using selection sort."

Specifically: The machine has
a 1-dimensional array
containing $\Theta(n)$ "cells."
Each cell stores $\Theta(\lg n)$ bits.

Input and output are
stored in these cells.

The machine also has a
"head" moving through array.
Head contains $\Theta(1)$ cells.

Head can see the cell at
its current array position;
perform arithmetic etc.;
move to adjacent array position.

Selection sort: Head
looks at each array position,
picks up the largest number,
moves it to the end of the array,
picks up the second largest,
etc.

ds on
works.

machine is a
achine
t."

machine has
rray
"cells."
$(\lg n)$ bits.

are
ls.

The machine also has a
"head" moving through array.
Head contains $\Theta(1)$ cells.

Head can see the cell at
its current array position;
perform arithmetic etc.;
move to adjacent array position.

Selection sort: Head
looks at each array position,
picks up the largest number,
moves it to the end of the array,
picks up the second largest,
etc.

Moving to adjacen
takes $n^{o(1)}$ second

Moving a number
takes $n^{1+o(1)}$ seco
Same for comparis

Total sorting time
$n^{2+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ Euros
for $n^{1+o(1)}$ cells.
Negligible extra co

The machine also has a
"head" moving through array.
Head contains $\Theta(1)$ cells.

Head can see the cell at
its current array position;
perform arithmetic etc.;
move to adjacent array position.

Selection sort: Head
looks at each array position,
picks up the largest number,
moves it to the end of the array,
picks up the second largest,
etc.

Moving to adjacent array position
takes $n^{o(1)}$ seconds.

Moving a number to end of array
takes $n^{1+o(1)}$ seconds.
Same for comparisons etc.

Total sorting time:
$n^{2+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ Euros
for $n^{1+o(1)}$ cells.
Negligible extra cost for head.

has a
rough array.
1) cells.

cell at
osition;
c etc.;
array position.

ad
y position,
st number,
d of the array,
nd largest,

Moving to adjacent array position takes $n^{o(1)}$ seconds.

Moving a number to end of array takes $n^{1+o(1)}$ seconds.
Same for comparisons etc.

Total sorting time:
$n^{2+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ Euros
for $n^{1+o(1)}$ cells.
Negligible extra cost for head.

Possibility 2: The
"2-dimensional RA
using merge sort."

Machine has $\Theta(n)$
in a 2-dimensional
$\Theta(\sqrt{n})$ rows, $\Theta(\sqrt{\ }$
Machine also has a

Merge sort: Head
sorts first $\lfloor n/2 \rfloor$ n
sorts last $\lceil n/2 \rceil$ nu
merges the sorted

Moving to adjacent array position takes $n^{o(1)}$ seconds.

Moving a number to end of array takes $n^{1+o(1)}$ seconds.
Same for comparisons etc.

Total sorting time:
$n^{2+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ Euros
for $n^{1+o(1)}$ cells.
Negligible extra cost for head.

Possibility 2: The machine is a "2-dimensional RAM using merge sort."

Machine has $\Theta(n)$ cells
in a 2-dimensional array:
$\Theta(\sqrt{n})$ rows, $\Theta(\sqrt{n})$ columns.
Machine also has a head.

Merge sort: Head recursively
sorts first $\lfloor n/2 \rfloor$ numbers;
sorts last $\lceil n/2 \rceil$ numbers;
merges the sorted lists.

nt array position
ls.

to end of array
onds.
sons etc.
:

ost for head.

Possibility 2: The machine is a “2-dimensional RAM using merge sort.”

Machine has $\Theta(n)$ cells in a 2-dimensional array: $\Theta(\sqrt{n})$ rows, $\Theta(\sqrt{n})$ columns. Machine also has a head.

Merge sort: Head recursively sorts first $\lfloor n/2 \rfloor$ numbers; sorts last $\lceil n/2 \rceil$ numbers; merges the sorted lists.

Merging requires n
to “random” array

Average jump: n^0
to adjacent array p

Each move takes n

Total sorting time:
$n^{1.5+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ Euros.

Possibility 2: The machine is a "2-dimensional RAM using merge sort."

Machine has $\Theta(n)$ cells in a 2-dimensional array: $\Theta(\sqrt{n})$ rows, $\Theta(\sqrt{n})$ columns. Machine also has a head.

Merge sort: Head recursively sorts first $\lfloor n/2 \rfloor$ numbers; sorts last $\lceil n/2 \rceil$ numbers; merges the sorted lists.

Merging requires $n^{1+o(1)}$ jumps to "random" array positions.

Average jump: $n^{0.5+o(1)}$ moves to adjacent array positions.

Each move takes $n^{o(1)}$ seconds.

Total sorting time: $n^{1.5+o(1)}$ seconds.

Cost of machine: once again $n^{1+o(1)}$ Euros.

machine is a
AM

) cells
array:
$\sqrt{n})$ columns.
a head.

recursively
umbers;
umbers;
lists.

Merging requires $n^{1+o(1)}$ jumps to "random" array positions.

Average jump: $n^{0.5+o(1)}$ moves to adjacent array positions.

Each move takes $n^{o(1)}$ seconds.

Total sorting time:
$n^{1.5+o(1)}$ seconds.

Cost of machine: once again
$n^{1+o(1)}$ Euros.

Possibility 3: The
"pipelined 2-dimen
using radix-2 sort."

Machine has $\Theta(n)$
in a 2-dimensional
Each cell in the ar
network links to th
cells in the same c
Each cell in the to
network links to th
cells in the top row

Machine also has a
attached to top-lef

Merging requires $n^{1+o(1)}$ jumps to "random" array positions.

Average jump: $n^{0.5+o(1)}$ moves to adjacent array positions.

Each move takes $n^{o(1)}$ seconds.

Total sorting time:
$n^{1.5+o(1)}$ seconds.

Cost of machine: once again
$n^{1+o(1)}$ Euros.

Possibility 3: The machine is a "pipelined 2-dimensional RAM using radix-2 sort."

Machine has $\Theta(n)$ cells
in a 2-dimensional array.
Each cell in the array has
network links to the 2 adjacent
cells in the same column.
Each cell in the top row has
network links to the 2 adjacent
cells in the top row.

Machine also has a CPU
attached to top-left cell.

$n^{1+o(1)}$ jumps
positions.

$.5+o(1)$ moves
positions.

$n^{o(1)}$ seconds.

once again

Possibility 3: The machine is a "pipelined 2-dimensional RAM using radix-2 sort."

Machine has $\Theta(n)$ cells in a 2-dimensional array.
Each cell in the array has network links to the 2 adjacent cells in the same column.
Each cell in the top row has network links to the 2 adjacent cells in the top row.

Machine also has a CPU attached to top-left cell.

Radix-2 sort: CPU
shuffles array using
even numbers befo
3 1 4 1 5 9 2 6 $\mapsto$
4 2 6 3 1 1 5 9.

Then using bit 1:
4 1 1 5 9 2 6 3.

Then using bit 2:
1 1 9 2 3 4 5 6.

Then using bit 3:
1 1 2 3 4 5 6 9.

etc. $\Theta(\lg n)$ bits.

Possibility 3: The machine is a "pipelined 2-dimensional RAM using radix-2 sort."

Machine has $\Theta(n)$ cells in a 2-dimensional array.
Each cell in the array has network links to the 2 adjacent cells in the same column.
Each cell in the top row has network links to the 2 adjacent cells in the top row.

Machine also has a CPU attached to top-left cell.

Radix-2 sort: CPU shuffles array using bit 0, even numbers before odd.
3 1 4 1 5 9 2 6 $\mapsto$
4 2 6 3 1 1 5 9.

Then using bit 1:
4 1 1 5 9 2 6 3.

Then using bit 2:
1 1 9 2 3 4 5 6.

Then using bit 3:
1 1 2 3 4 5 6 9.

etc. $\Theta(\lg n)$ bits.

machine is a
nsional RAM
”

) cells
array.
ray has
ne 2 adjacent
column.
op row has
ne 2 adjacent
w.

a CPU
ft cell.

Radix-2 sort: CPU shuffles array using bit 0, even numbers before odd.
3 1 4 1 5 9 2 6 $\mapsto$
4 2 6 3 1 1 5 9.

Then using bit 1:
4 1 1 5 9 2 6 3.

Then using bit 2:
1 1 9 2 3 4 5 6.

Then using bit 3:
1 1 2 3 4 5 6 9.

etc. $\Theta(\lg n)$ bits.

CPU can read/wri
sending request th
Does not need to
before sending nex

CPU can read an
of $n^{0.5+o(1)}$ cells
in $n^{0.5+o(1)}$ secon
Sends all requests,
then receives respo

Total sorting time
$n^{1+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ Euros.

Radix-2 sort: CPU shuffles array using bit 0, even numbers before odd.

3 1 4 1 5 9 2 6 $\mapsto$
4 2 6 3 1 1 5 9.

Then using bit 1:
4 1 1 5 9 2 6 3.

Then using bit 2:
1 1 9 2 3 4 5 6.

Then using bit 3:
1 1 2 3 4 5 6 9.

etc. $\Theta(\lg n)$ bits.

CPU can read/write any cell by sending request through network. Does not need to wait for response before sending next request.

CPU can read an entire row of $n^{0.5+o(1)}$ cells in $n^{0.5+o(1)}$ seconds. Sends all requests, then receives responses.

Total sorting time: $n^{1+o(1)}$ seconds.

Cost of machine: once again $n^{1+o(1)}$ Euros.

U
g bit 0,
ore odd.

CPU can read/write any cell by
sending request through network.
Does not need to wait for response
before sending next request.

CPU can read an entire row
of $n^{0.5+o(1)}$ cells
in $n^{0.5+o(1)}$ seconds.
Sends all requests,
then receives responses.

Total sorting time:
$n^{1+o(1)}$ seconds.

Cost of machine: once again
$n^{1+o(1)}$ Euros.

Possibility 4: The
"2-dimensional me
using Schimmler s

Machine has $\Theta(n)$
in a 2-dimensional
Each cell has netw
to the 4 adjacent

Machine also has
attached to top-le
CPU broadcasts in
to all of the cells,
cells do most of th

CPU can read/write any cell by sending request through network. Does not need to wait for response before sending next request.

CPU can read an entire row of $n^{0.5+o(1)}$ cells in $n^{0.5+o(1)}$ seconds. Sends all requests, then receives responses.

Total sorting time: $n^{1+o(1)}$ seconds.

Cost of machine: once again $n^{1+o(1)}$ Euros.

Possibility 4: The machine is a "2-dimensional mesh using Schimmler sort."

Machine has $\Theta(n)$ cells in a 2-dimensional array. Each cell has network links to the 4 adjacent cells.

Machine also has a CPU attached to top-left cell. CPU broadcasts instructions to all of the cells, but cells do most of the processing.

te any cell by
rough network.
wait for response
xt request.

entire row

ds.

onses.

:

once again

Possibility 4: The machine is a "2-dimensional mesh using Schimmler sort."

Machine has $\Theta(n)$ cells in a 2-dimensional array. Each cell has network links to the 4 adjacent cells.

Machine also has a CPU attached to top-left cell. CPU broadcasts instructions to all of the cells, but cells do most of the processing.

Schimmler sort:
Recursively sort qu
in parallel. Then f
Sort each column
Sort each row in p
Sort each column
Sort each row in p

With proper choic
left-to-right/right-
for each row, can
that this sorts who

Possibility 4: The machine is a "2-dimensional mesh using Schimmler sort."

Machine has $\Theta(n)$ cells in a 2-dimensional array. Each cell has network links to the 4 adjacent cells.

Machine also has a CPU attached to top-left cell. CPU broadcasts instructions to all of the cells, but cells do most of the processing.

Schimmler sort:
Recursively sort quadrants in parallel. Then four steps:
Sort each column in parallel.
Sort each row in parallel.
Sort each column in parallel.
Sort each row in parallel.

With proper choice of left-to-right/right-to-left for each row, can prove that this sorts whole array.

machine is a
esh
ort."

) cells
array.
ork links
cells.

a CPU
ft cell.
nstructions
but
ne processing.

Schimmler sort:

Recursively sort quadrants in parallel. Then four steps:

Sort each column in parallel.

Sort each row in parallel.

Sort each column in parallel.

Sort each row in parallel.

With proper choice of left-to-right/right-to-left for each row, can prove that this sorts whole array.

To sort one row:

Sort each pair in p

3 1 4 1 5 9 2 6 ↦

1 3 1 4 5 9 2 6

Sort alternate pair

1 3 1 4 5 9 2 6 ↦

1 1 3 4 5 2 9 6

Repeat.

Can prove that ro
when number of st
equals row length.

Schimmler sort:

Recursively sort quadrants in parallel. Then four steps:

Sort each column in parallel.

Sort each row in parallel.

Sort each column in parallel.

Sort each row in parallel.

With proper choice of left-to-right/right-to-left for each row, can prove that this sorts whole array.

To sort one row:

Sort each pair in parallel.

$\underline{3\ 1}\ \underline{4\ 1}\ \underline{5\ 9}\ \underline{2\ 6} \mapsto$

$1\ 3\ 1\ 4\ 5\ 9\ 2\ 6$

Sort alternate pairs in parallel.

$1\ \underline{3\ 1}\ \underline{4\ 5}\ \underline{9\ 2}\ 6 \mapsto$

$1\ 1\ 3\ 4\ 5\ 2\ 9\ 6$

Repeat.

Can prove that row is sorted when number of steps equals row length.

uadrants
our steps:
in parallel.
parallel.
in parallel.
parallel.

e of
to-left
prove
ble array.

To sort one row:

Sort each pair in parallel.

3 1 4 1 5 9 2 6 $\mapsto$
1 3 1 4 5 9 2 6

Sort alternate pairs in parallel.

1 3 1 4 5 9 2 6 $\mapsto$
1 1 3 4 5 2 9 6

Repeat.

Can prove that row is sorted
when number of steps
equals row length.

Sort one row
in $n^{0.5+o(1)}$ secon

All rows in parallel
$n^{0.5+o(1)}$ seconds.

Total sorting time
$n^{0.5+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ Euros.

To sort one row:

Sort each pair in parallel.

<u>3 1</u> <u>4 1</u> <u>5 9</u> <u>2 6</u> $\mapsto$

1 3 1 4 5 9 2 6

Sort alternate pairs in parallel.

1 <u>3 1</u> <u>4 5</u> <u>9 2</u> 6 $\mapsto$

1 1 3 4 5 2 9 6

Repeat.

Can prove that row is sorted when number of steps equals row length.

Sort one row in $n^{0.5+o(1)}$ seconds.

All rows in parallel: $n^{0.5+o(1)}$ seconds.

Total sorting time: $n^{0.5+o(1)}$ seconds.

Cost of machine: once again $n^{1+o(1)}$ Euros.

parallel.

s in parallel.

w is sorted
teps

Sort one row
in $n^{0.5+o(1)}$ seconds.

All rows in parallel:
$n^{0.5+o(1)}$ seconds.

Total sorting time:
$n^{0.5+o(1)}$ seconds.

Cost of machine: once again
$n^{1+o(1)}$ Euros.

Some philosophica

1-tape Turing mac
RAMs, 2-dimensio
compute the same

Prove this by prov
each machine can
computations on t

(We believe that e
reasonable model
can be simulated b
1-tape Turing mac
“Church-Turing th

Sort one row
in $n^{0.5+o(1)}$ seconds.

All rows in parallel:
$n^{0.5+o(1)}$ seconds.

Total sorting time:
$n^{0.5+o(1)}$ seconds.

Cost of machine: once again
$n^{1+o(1)}$ Euros.

Some philosophical notes

1-tape Turing machines,
RAMs, 2-dimensional meshes
compute the same functions.

Prove this by proving that
each machine can simulate
computations on the others.

(We believe that *every*
reasonable model of computation
can be simulated by a
1-tape Turing machine.
"Church-Turing thesis.")

ds.

l:

:

once again

Some philosophical notes

1-tape Turing machines, RAMs, 2-dimensional meshes compute the same functions.

Prove this by proving that each machine can simulate computations on the others.

(We believe that *every* reasonable model of computation can be simulated by a 1-tape Turing machine. "Church-Turing thesis.")

1-tape Turing mac
RAMs, 2-dimensio
compute the same
in polynomial time
at polynomial cost

Prove this by prov
simulations are po

(Is this true for ev
reasonable model
Consider quantum

Some philosophical notes

1-tape Turing machines, RAMs, 2-dimensional meshes compute the same functions.

Prove this by proving that each machine can simulate computations on the others.

(We believe that *every* reasonable model of computation can be simulated by a 1-tape Turing machine. "Church-Turing thesis.")

1-tape Turing machines, RAMs, 2-dimensional meshes compute the same functions in polynomial time at polynomial cost.

Prove this by proving that simulations are polynomial.

(Is this true for every reasonable model of computation? Consider quantum computers.)

l notes

chines,
nal meshes
functions.

ing that
simulate
he others.

every
of computation
y a
chine.
esis.”)

1-tape Turing machines,
RAMs, 2-dimensional meshes
compute the same functions
in polynomial time
at polynomial cost.

Prove this by proving that
simulations are polynomial.

(Is this true for every
reasonable model of computation?
Consider quantum computers.)

1-tape Turing mac
RAMs, 2-dimensio
do not compute
the same function
within, e.g., time
and cost $n^{1+o(1)}$.

Example: 1-tape
cannot sort in tim
Too local!

Example: 2-dimen
cannot sort in tim
Too sequential!

1-tape Turing machines,
RAMs, 2-dimensional meshes
compute the same functions
in polynomial time
at polynomial cost.

Prove this by proving that
simulations are polynomial.

(Is this true for every
reasonable model of computation?
Consider quantum computers.)

1-tape Turing machines,
RAMs, 2-dimensional meshes
do not compute
the same functions
within, e.g., time $n^{1+o(1)}$
and cost $n^{1+o(1)}$.

Example: 1-tape Turing machine
cannot sort in time $n^{1+o(1)}$.
Too local!

Example: 2-dimensional RAM
cannot sort in time $n^{0.5+o(1)}$.
Too sequential!

chines,
nal meshes
e functions
e
t.

ing that
lynomial.

ery
of computation?
computers.)

1-tape Turing machines,
RAMs, 2-dimensional meshes
do not compute
the same functions
within, e.g., time $n^{1+o(1)}$
and cost $n^{1+o(1)}$.

Example: 1-tape Turing machine
cannot sort in time $n^{1+o(1)}$.
Too local!

Example: 2-dimensional RAM
cannot sort in time $n^{0.5+o(1)}$.
Too sequential!

$o(1)$ is asymptotic
Speedup factor suc
might not be a spe
for small values of

When n is small,
RAM might seem
sensible machine c

But, for large n,
having a huge mem
waiting for a single
is a silly machine c

1-tape Turing machines,
RAMs, 2-dimensional meshes
do not compute
the same functions
within, e.g., time $n^{1+o(1)}$
and cost $n^{1+o(1)}$.

Example: 1-tape Turing machine
cannot sort in time $n^{1+o(1)}$.
Too local!

Example: 2-dimensional RAM
cannot sort in time $n^{0.5+o(1)}$.
Too sequential!

$o(1)$ is asymptotic.
Speedup factor such as $n^{0.5+o(1)}$
might not be a speedup
for small values of n.

When n is small,
RAM might seem to be a
sensible machine design.

But, for large n,
having a huge memory
waiting for a single CPU
is a silly machine design.

chines,
nal meshes

s
$n^{1+o(1)}$

Turing machine
e $n^{1+o(1)}$.

sional RAM
e $n^{0.5+o(1)}$.

$o(1)$ is asymptotic.
Speedup factor such as $n^{0.5+o(1)}$
might not be a speedup
for small values of n.

When n is small,
RAM might seem to be a
sensible machine design.

But, for large n,
having a huge memory
waiting for a single CPU
is a silly machine design.

Myth:
Parallel computati
improve price-perf
p parallel compute
may reduce time b
but increase cost b

Reality: Can often
a *large* serial comp
into p *small* parall
so cost does *not*
increase by factor

$o(1)$ is asymptotic.
Speedup factor such as $n^{0.5+o(1)}$
might not be a speedup
for small values of n.

When n is small,
RAM might seem to be a
sensible machine design.

But, for large n,
having a huge memory
waiting for a single CPU
is a silly machine design.

Myth:
Parallel computation cannot
improve price-performance ratio;
p parallel computers
may reduce time by factor p
but increase cost by factor p.

Reality: Can often convert
a *large* serial computer
into p *small* parallel cells,
so cost does *not*
increase by factor p.

.
ch as $n^{0.5+o(1)}$
eedup
n.

to be a
lesign.

mory
e CPU
design.

Myth:
Parallel computation cannot
improve price-performance ratio;
p parallel computers
may reduce time by factor p
but increase cost by factor p.

Reality: Can often convert
a *large* serial computer
into p *small* parallel cells,
so cost does *not*
increase by factor p.

Myth: Designing a
cannot produce m
small constant-fac
compared to, e.g.,
What matters is s
streamlining, such
instruction-decodi

Reality: In 1997, D
was 1000 times fa
set of Pentiums at
What matters is p

Myth:
Parallel computation cannot improve price-performance ratio; p parallel computers may reduce time by factor p but increase cost by factor p.

Reality: Can often convert a *large* serial computer into p *small* parallel cells, so cost does *not* increase by factor p.

Myth: Designing a new machine cannot produce more than a small constant-factor improvement compared to, e.g., a Pentium. What matters is special-purpose streamlining, such as reducing instruction-decoding costs.

Reality: In 1997, DES Cracker was 1000 times faster than a set of Pentiums at the same price. What matters is parallelism.

on cannot
ormance ratio;
ers
by factor p
by factor p.

n convert
puter
el cells,

p.

Myth: Designing a new machine cannot produce more than a small constant-factor improvement compared to, e.g., a Pentium. What matters is special-purpose streamlining, such as reducing instruction-decoding costs.

Reality: In 1997, DES Cracker was 1000 times faster than a set of Pentiums at the same price. What matters is parallelism.

Future computers
massively parallel

Computer designer
today's RAM-style
just as we laugh a
a 1-tape Turing m

Algorithm experts
today's dominant
algorithm analysis,
count CPU "opera
view memory acce

Myth: Designing a new machine cannot produce more than a small constant-factor improvement compared to, e.g., a Pentium. What matters is special-purpose streamlining, such as reducing instruction-decoding costs.

Reality: In 1997, DES Cracker was 1000 times faster than a set of Pentiums at the same price. What matters is parallelism.

Future computers will be massively parallel meshes.

Computer designers will laugh at today's RAM-style machines, just as we laugh at a 1-tape Turing machine.

Algorithm experts will laugh at today's dominant style of algorithm analysis, where we count CPU "operations" and view memory access as free.

a new machine
ore than a
tor improvement
a Pentium.
pecial-purpose
as reducing
ng costs.

DES Cracker
ster than a
t the same price.
arallelism.

Future computers will be massively parallel meshes.

Computer designers will laugh at today's RAM-style machines, just as we laugh at a 1-tape Turing machine.

Algorithm experts will laugh at today's dominant style of algorithm analysis, where we count CPU "operations" and view memory access as free.

Brute-force search

For each 128-bit A
define $H(k) =$ AE

Typical known-pla
given $H(k)$; want

Cryptanalyst build
p parallel AES circ
each guessing n k
for a total of pn k

Time: n AES eval
Cost: p AES circu
Success chance: p

Future computers will be massively parallel meshes.

Computer designers will laugh at today's RAM-style machines, just as we laugh at a 1-tape Turing machine.

Algorithm experts will laugh at today's dominant style of algorithm analysis, where we count CPU "operations" and view memory access as free.

Brute-force searches

For each 128-bit AES key k define $H(k) = \text{AES}_k(0)$.

Typical known-plaintext attack: given $H(k)$; want to find k.

Cryptanalyst builds machine with p parallel AES circuits, each guessing n keys, for a total of pn keys.

Time: n AES evaluations.
Cost: p AES circuits.
Success chance: $pn/2^{128}$.

will be
meshes.

rs will laugh at
machines,
t
achine.

will laugh at
style of
, where we
ations" and
ss as free.

Brute-force searches

For each 128-bit AES key k define $H(k) = \text{AES}_k(0)$.

Typical known-plaintext attack: given $H(k)$; want to find k.

Cryptanalyst builds machine with p parallel AES circuits, each guessing n keys, for a total of pn keys.

Time: n AES evaluations.
Cost: p AES circuits.
Success chance: $pn/2^{128}$.

Cryptanalyst is act
attacking many AE
Wants to find k_1,
given $H(k_1), H(k_2$

Rivest's "time-mer
using distinguished
merges these comp

For any 128-bit r:
$H(r), H(H(r)), \ldots$
finding string that
with 30 zero bits.
Call that string $Z($

Brute-force searches

For each 128-bit AES key k define $H(k) = \text{AES}_k(0)$.

Typical known-plaintext attack: given $H(k)$; want to find k.

Cryptanalyst builds machine with p parallel AES circuits, each guessing n keys, for a total of pn keys.

Time: n AES evaluations.
Cost: p AES circuits.
Success chance: $pn/2^{128}$.

Cryptanalyst is actually attacking many AES keys. Wants to find $k_1, k_2, \ldots$ given $H(k_1), H(k_2), \ldots$.

Rivest's "time-memory tradeoff using distinguished points" merges these computations.

For any 128-bit r: Compute $H(r), H(H(r)), \ldots$ until finding string that begins with 30 zero bits.
Call that string $Z(r)$.

es

AES key k
$S_k(0)$.

intext attack:
to find k.

s machine with
cuits,
eys,
keys.

luations.
its.
$n/2^{128}$.

Cryptanalyst is actually attacking many AES keys.
Wants to find $k_1, k_2, \ldots$
given $H(k_1), H(k_2), \ldots$.

Rivest's "time-memory tradeoff using distinguished points" merges these computations.

For any 128-bit r: Compute $H(r), H(H(r)), \ldots$ until finding string that begins with 30 zero bits.
Call that string $Z(r)$.

Given $H(k_1), H(k_2$

Choose random r_1
Store $Z(r_1), Z(r_2$
in an array in RAM

Compute each $Z($
look up $Z(H(k_i))$

If $Z(H(k_i)) = Z($
check whether $H($
any of $H(r_j), H(H$

Details: avoid infi
handle multiple co

Cryptanalyst is actually attacking many AES keys. Wants to find $k_1, k_2, \ldots$ given $H(k_1), H(k_2), \ldots$.

Rivest's "time-memory tradeoff using distinguished points" merges these computations.

For any 128-bit r: Compute $H(r), H(H(r)), \ldots$ until finding string that begins with 30 zero bits. Call that string $Z(r)$.

Given $H(k_1), H(k_2), \ldots, H(k_p)$:

Choose random $r_1, r_2, \ldots, r_p$. Store $Z(r_1), Z(r_2), \ldots, Z(r_p)$ in an array in RAM.

Compute each $Z(H(k_i))$; look up $Z(H(k_i))$ in the array.

If $Z(H(k_i)) = Z(r_j)$, check whether $H(k_i)$ matches any of $H(r_j), H(H(r_j)), \ldots$.

Details: avoid infinite loops; handle multiple collisions.

tually
ES keys.
$k_2, \ldots$
$), \ldots$.

mory tradeoff
d points"
putations.

Compute
. until
begins

(r).

Given $H(k_1), H(k_2), \ldots, H(k_p)$:

Choose random $r_1, r_2, \ldots, r_p$.
Store $Z(r_1), Z(r_2), \ldots, Z(r_p)$
in an array in RAM.

Compute each $Z(H(k_i))$;
look up $Z(H(k_i))$ in the array.

If $Z(H(k_i)) = Z(r_j)$,
check whether $H(k_i)$ matches
any of $H(r_j), H(H(r_j)), \ldots$.

Details: avoid infinite loops;
handle multiple collisions.

Heuristic analysis:
$Z(r_1), Z(r_2), \ldots,$
involves $\approx 2^{30}$ out

If any of the input
then we'll find k_1.
Chance $\approx 2^{30}p/2^1$

Same for $k_2, k_3, \ldots$
Total chance $\approx 2^3$
of finding at least

On a *serial* compu
$\approx 2^{31}p$ AES evalu
Cost: $\approx 128p$ bits

Given $H(k_1), H(k_2), \ldots, H(k_p)$:

Choose random $r_1, r_2, \ldots, r_p$.
Store $Z(r_1), Z(r_2), \ldots, Z(r_p)$
in an array in RAM.

Compute each $Z(H(k_i))$;
look up $Z(H(k_i))$ in the array.

If $Z(H(k_i)) = Z(r_j)$,
check whether $H(k_i)$ matches
any of $H(r_j), H(H(r_j)), \ldots$.

Details: avoid infinite loops;
handle multiple collisions.

Heuristic analysis: Computing
$Z(r_1), Z(r_2), \ldots, Z(r_p)$
involves $\approx 2^{30}$ outputs of H.

If any of the inputs match k_1
then we'll find k_1.
Chance $\approx 2^{30}p/2^{128}$.

Same for $k_2, k_3, \ldots$.
Total chance $\approx 2^{30}p^2/2^{128}$
of finding at least one key.

On a *serial* computer,
$\approx 2^{31}p$ AES evaluations.
Cost: $\approx 128p$ bits of memory.

$_2), \ldots, H(k_p)$:

$, r_2, \ldots, r_p$.

$), \ldots, Z(r_p)$

M.

$H(k_i))$;

in the array.

$r_j)$,

$k_i)$ matches

$H(r_j)), \ldots$.

nite loops;

llisions.

Heuristic analysis: Computing $Z(r_1), Z(r_2), \ldots, Z(r_p)$ involves $\approx 2^{30}$ outputs of H.

If any of the inputs match k_1 then we'll find k_1.
Chance $\approx 2^{30}p/2^{128}$.

Same for $k_2, k_3, \ldots$.
Total chance $\approx 2^{30}p^2/2^{128}$ of finding at least one key.

On a *serial* computer, $\approx 2^{31}p$ AES evaluations.
Cost: $\approx 128p$ bits of memory.

Much better: Mas

Compute all Z val
using p AES circui
Use Schimmler sor
collisions $Z(H(k_i)$

Time: $\approx 2^{31}$ AES
plus $\approx 8\sqrt{p}$ Schim
About p times fast

Cost: p AES circu
plus network links.
Maybe 100 times
than serial. Can re

Heuristic analysis: Computing $Z(r_1), Z(r_2), \ldots, Z(r_p)$ involves $\approx 2^{30}$ outputs of H.

If any of the inputs match k_1 then we'll find k_1. Chance $\approx 2^{30}p/2^{128}$.

Same for $k_2, k_3, \ldots$. Total chance $\approx 2^{30}p^2/2^{128}$ of finding at least one key.

On a *serial* computer, $\approx 2^{31}p$ AES evaluations. Cost: $\approx 128p$ bits of memory.

Much better: Massive parallelism.

Compute all Z values in parallel, using p AES circuits. Use Schimmler sort to find collisions $Z(H(k_i)) = Z(r_j)$.

Time: $\approx 2^{31}$ AES evaluations, plus $\approx 8\sqrt{p}$ Schimmler steps. About p times faster than serial.

Cost: p AES circuits, plus network links. Maybe 100 times more expensive than serial. Can reduce the 100.

Computing
$Z(r_p)$
tputs of H.

s match k_1

^{128}.

..
$^{0}p^2/2^{128}$
one key.

uter,
ations.
of memory.

Much better: Massive parallelism.

Compute all Z values in parallel, using p AES circuits.
Use Schimmler sort to find collisions $Z(H(k_i)) = Z(r_j)$.

Time: $\approx 2^{31}$ AES evaluations, plus $\approx 8\sqrt{p}$ Schimmler steps.
About p times faster than serial.

Cost: p AES circuits, plus network links.
Maybe 100 times more expensive than serial. Can reduce the 100.

Sieving

The "number-field
is today's fastest r
to factor a big RS

Most important N
find small prime di
of $x, x+1, x+2,$

1000002: divisible
1000003:
1000004: divisible
1000005: divisible
1000006: divisible

Much better: Massive parallelism.

Compute all Z values in parallel, using p AES circuits. Use Schimmler sort to find collisions $Z(H(k_i)) = Z(r_j)$.

Time: $\approx 2^{31}$ AES evaluations, plus $\approx 8\sqrt{p}$ Schimmler steps. About p times faster than serial.

Cost: p AES circuits, plus network links. Maybe 100 times more expensive than serial. Can reduce the 100.

Sieving

The "number-field sieve" (NFS) is today's fastest method to factor a big RSA key n.

Most important NFS bottleneck: find small prime divisors of $x, x+1, x+2, \ldots, x+y$.

1000002: divisible by 2 3
1000003:
1000004: divisible by 2 2
1000005: divisible by 3 5
1000006: divisible by 2 7

ssive parallelism.

ues in parallel,
its.
rt to find
$) = Z(r_j)$.

evaluations,
mmler steps.
ter than serial.

its,
.
more expensive
educe the 100.

Sieving

The "number-field sieve" (NFS) is today's fastest method to factor a big RSA key n.

Most important NFS bottleneck: find small prime divisors of $x, x+1, x+2, \ldots, x+y$.

1000002: divisible by 2 3
1000003:
1000004: divisible by 2 2
1000005: divisible by 3 5
1000006: divisible by 2 7

Conventional sievi
(e.g. 2000 Silverm
2000 Lenstra Shar

Generate pairs (2,
$(2, 1000004)$, $(2, 1$
$(3, 1000002)$, $(3, 1$
etc.
Use distribution so
to sort by second

$y^{1+o(1)}$ pairs.
Sorting time $y^{1+o($
machine cost y^{1+o}

Sieving

The "number-field sieve" (NFS) is today's fastest method to factor a big RSA key n.

Most important NFS bottleneck: find small prime divisors of $x, x+1, x+2, \ldots, x+y$.

1000002: divisible by 2 3
1000003:
1000004: divisible by 2 2
1000005: divisible by 3 5
1000006: divisible by 2 7

Conventional sieving/TWINKLE (e.g. 2000 Silverman, 2000 Lenstra Shamir):

Generate pairs $(2, 1000002)$, $(2, 1000004)$, $(2, 1000006)$, $\ldots$, $(3, 1000002)$, $(3, 1000005)$, $\ldots$, etc.
Use distribution sort to sort by second component.

$y^{1+o(1)}$ pairs.
Sorting time $y^{1+o(1)}$;
machine cost $y^{1+o(1)}$.

l sieve" (NFS)
method
A key n.

FS bottleneck:
ivisors
$\ldots, x+y$.

by 2 3

by 2 2
by 3 5
by 2 7

Conventional sieving/TWINKLE
(e.g. 2000 Silverman,
2000 Lenstra Shamir):

Generate pairs $(2, 1000002)$,
$(2, 1000004)$, $(2, 1000006)$, $\ldots$,
$(3, 1000002)$, $(3, 1000005)$, $\ldots$,
etc.
Use distribution sort
to sort by second component.

$y^{1+o(1)}$ pairs.
Sorting time $y^{1+o(1)}$;
machine cost $y^{1+o(1)}$.

For same machine
achieve much high
by massive paralle

e.g. Schimmler so
sorting time $y^{0.5+}$
machine cost y^{1+}

This drastically re
overall NFS time
for sufficiently larg
(2001 Bernstein)

Conventional sieving/TWINKLE (e.g. 2000 Silverman, 2000 Lenstra Shamir):

Generate pairs $(2, 1000002)$, $(2, 1000004)$, $(2, 1000006)$, $\ldots$, $(3, 1000002)$, $(3, 1000005)$, $\ldots$, etc.
Use distribution sort to sort by second component.

$y^{1+o(1)}$ pairs.
Sorting time $y^{1+o(1)}$;
machine cost $y^{1+o(1)}$.

For same machine cost, achieve much higher speed by massive parallelism.

e.g. Schimmler sort:
sorting time $y^{0.5+o(1)}$;
machine cost $y^{1+o(1)}$.

This drastically reduces overall NFS time for sufficiently large n.
(2001 Bernstein)

ng/TWINKLE
an,
mir):

1000002),
000006), . . . ,
000005), . . . ,

ort
component.

(1);
$o(1)$.

For same machine cost,
achieve much higher speed
by massive parallelism.

e.g. Schimmler sort:
sorting time $y^{0.5+o(1)}$;
machine cost $y^{1+o(1)}$.

This drastically reduces
overall NFS time
for sufficiently large n.
(2001 Bernstein)

Can do even bette
low-memory small-
algorithms, such a
elliptic-curve meth
Time only $y^{0+o(1)}$
machine cost y^{1+o}

This further reduc
overall NFS time
for sufficiently larg
(2001 Bernstein)

Can also save time
bottleneck, "linear
less important. (2

For same machine cost, achieve much higher speed by massive parallelism.

e.g. Schimmler sort: sorting time $y^{0.5+o(1)}$; machine cost $y^{1+o(1)}$.

This drastically reduces overall NFS time for sufficiently large n. (2001 Bernstein)

Can do even better with low-memory small-divisor algorithms, such as the elliptic-curve method (ECM).

Time only $y^{0+o(1)}$; machine cost $y^{1+o(1)}$.

This further reduces overall NFS time for sufficiently large n. (2001 Bernstein)

Can also save time in another bottleneck, "linear algebra"; less important. (2001 Bernstein)

cost,
ner speed
lism.

rt:
$o(1)$;
$o(1)$.

duces

ge n.

Can do even better with low-memory small-divisor algorithms, such as the elliptic-curve method (ECM).

Time only $y^{0+o(1)}$; machine cost $y^{1+o(1)}$.

This further reduces overall NFS time for sufficiently large n. (2001 Bernstein)

Can also save time in another bottleneck, “linear algebra”; less important. (2001 Bernstein)

NFS price-perform
$\exp((\beta+o(1))\sqrt[3]{(\mathrm{l}$
assuming standard

sieving	linear
RAM	RAM
RAM	RAM
Schimmler	RAM
Schimmler	Schim
ECM	RAM
ECM	Schim

(RAM 2.85: stand
2.37, 1.97: 2001.1
RAM 2.76: 2002.0

Can do even better with low-memory small-divisor algorithms, such as the elliptic-curve method (ECM).

Time only $y^{0+o(1)}$; machine cost $y^{1+o(1)}$.

This further reduces overall NFS time for sufficiently large n. (2001 Bernstein)

Can also save time in another bottleneck, "linear algebra"; less important. (2001 Bernstein)

NFS price-performance ratio is $\exp((\beta+o(1))\sqrt[3]{(\log n)(\log\log n)^2})$ assuming standard conjectures.

sieving	linear algebra	β
RAM	RAM	$2.85\ldots$
RAM	RAM	$2.76\ldots$
Schimmler	RAM	$2.37\ldots$
Schimmler	Schimmler	$2.36\ldots$
ECM	RAM	$2.08\ldots$
ECM	Schimmler	$1.97\ldots$

(RAM 2.85: standard;
2.37, 1.97: 2001.11 Bernstein;
RAM 2.76: 2002.04 Pomerance)

er with
-divisor
s the
od (ECM).

;
$o(1)$.

es

ge n.

e in another
r algebra";
001 Bernstein)

NFS price-performance ratio is
$\exp((\beta+o(1))\sqrt[3]{(\log n)(\log\log n)^2})$
assuming standard conjectures.

sieving	linear algebra	β
RAM	RAM	$2.85\ldots$
RAM	RAM	$2.76\ldots$
Schimmler	RAM	$2.37\ldots$
Schimmler	Schimmler	$2.36\ldots$
ECM	RAM	$2.08\ldots$
ECM	Schimmler	$1.97\ldots$

(RAM 2.85: standard;
2.37, 1.97: 2001.11 Bernstein;
RAM 2.76: 2002.04 Pomerance)

Switching from RA
massively parallel
produces gigantic
for sufficiently larg

Improvement from
RAM factorization
to best machine, β
corresponds to mu
number of digits o
by $3.009\ldots + o(1$

NFS price-performance ratio is $\exp((\beta+o(1))\sqrt[3]{(\log n)(\log\log n)^2})$ assuming standard conjectures.

sieving	linear algebra	β
RAM	RAM	$2.85\ldots$
RAM	RAM	$2.76\ldots$
Schimmler	RAM	$2.37\ldots$
Schimmler	Schimmler	$2.36\ldots$
ECM	RAM	$2.08\ldots$
ECM	Schimmler	$1.97\ldots$

(RAM 2.85: standard;
2.37, 1.97: 2001.11 Bernstein;
RAM 2.76: 2002.04 Pomerance)

Switching from RAM to a massively parallel machine produces gigantic NFS speedups for sufficiently large n.

Improvement from conventional RAM factorization, $\beta = 2.85\ldots$, to best machine, $\beta = 1.97\ldots$, corresponds to multiplying number of digits of n by $3.009\ldots + o(1)$.

ance ratio is

og n)(log log n)^2)

l conjectures.

algebra	β
	2.85 . . .
	2.76 . . .
	2.37 . . .
mler	2.36 . . .
	2.08 . . .
mler	1.97 . . .

lard;

1 Bernstein;

04 Pomerance)

Switching from RAM to a massively parallel machine produces gigantic NFS speedups for sufficiently large n.

Improvement from conventional RAM factorization, $\beta = 2.85\ldots$, to best machine, $\beta = 1.97\ldots$, corresponds to multiplying number of digits of n by $3.009\ldots + o(1)$.

As always, $o(1)$ is

Situation for small

is much less clear.

How expensive is i

factor 1024-bit RS

We still don't kno

Can now find man

making wild predic

None of the predic

can be taken serio

Switching from RAM to a massively parallel machine produces gigantic NFS speedups for sufficiently large n.

Improvement from conventional RAM factorization, $\beta = 2.85\ldots$, to best machine, $\beta = 1.97\ldots$, corresponds to multiplying number of digits of n by $3.009\ldots + o(1)$.

As always, $o(1)$ is asymptotic.

Situation for small n is much less clear.

How expensive is it to factor 1024-bit RSA keys? We still don't know.

Can now find many papers making wild predictions. None of the predictions can be taken seriously!

AM to a
machine
NFS speedups
ge n.

conventional
, $\beta = 2.85\ldots$,
$\beta = 1.97\ldots$,
ltiplying
f n
).

As always, $o(1)$ is asymptotic.

Situation for small n is much less clear.

How expensive is it to factor 1024-bit RSA keys? We still don't know.

Can now find many papers making wild predictions. None of the predictions can be taken seriously!

NFS speed is com

Example: NFS fac
using an auxiliary
Number of polyno
is huge. Effect of
takes time to com

Some papers don't
effort into polynon
so they underestim

Some papers make
optimal-polynomia
so they overestima

As always, $o(1)$ is asymptotic.

Situation for small n is much less clear.

How expensive is it to factor 1024-bit RSA keys? We still don't know.

Can now find many papers making wild predictions. None of the predictions can be taken seriously!

NFS speed is complicated.

Example: NFS factors n using an auxiliary polynomial. Number of polynomial choices is huge. Effect of polynomial takes time to compute.

Some papers don't put enough effort into polynomial choice, so they underestimate NFS speed.

Some papers make unjustified optimal-polynomial extrapolations, so they overestimate NFS speed.

asymptotic.

n

t to
SA keys?
w.

y papers
ctions.
ctions
usly!

NFS speed is complicated.

Example: NFS factors n using an auxiliary polynomial. Number of polynomial choices is huge. Effect of polynomial takes time to compute.

Some papers don't put enough effort into polynomial choice, so they underestimate NFS speed.

Some papers make unjustified optimal-polynomial extrapolations, so they overestimate NFS speed.

At a lower level, to
massively parallel c
are much less strea
than today's Penti

Computer market
Massive parallelism
become the de-fac
and will be tuned

How much speed w
Today it's hard to
But we'll find out!

NFS speed is complicated.

Example: NFS factors n using an auxiliary polynomial. Number of polynomial choices is huge. Effect of polynomial takes time to compute.

Some papers don't put enough effort into polynomial choice, so they underestimate NFS speed.

Some papers make unjustified optimal-polynomial extrapolations, so they overestimate NFS speed.

At a lower level, today's massively parallel computers are much less streamlined than today's Pentiums.

Computer market will evolve. Massive parallelism will become the de-facto standard, and will be tuned carefully.

How much speed will we gain? Today it's hard to say. But we'll find out!