Sieving Using Bucket Sort

Kazumaro Aoki Hiroki Ueda

NTT

This work was supported in part by TAO and by the CRYPTREC project.

Contents

- Memory characteristics in PC
- Sieving
- Bucket sort
- Proposed algorithm
- Numbers
- Conclusion

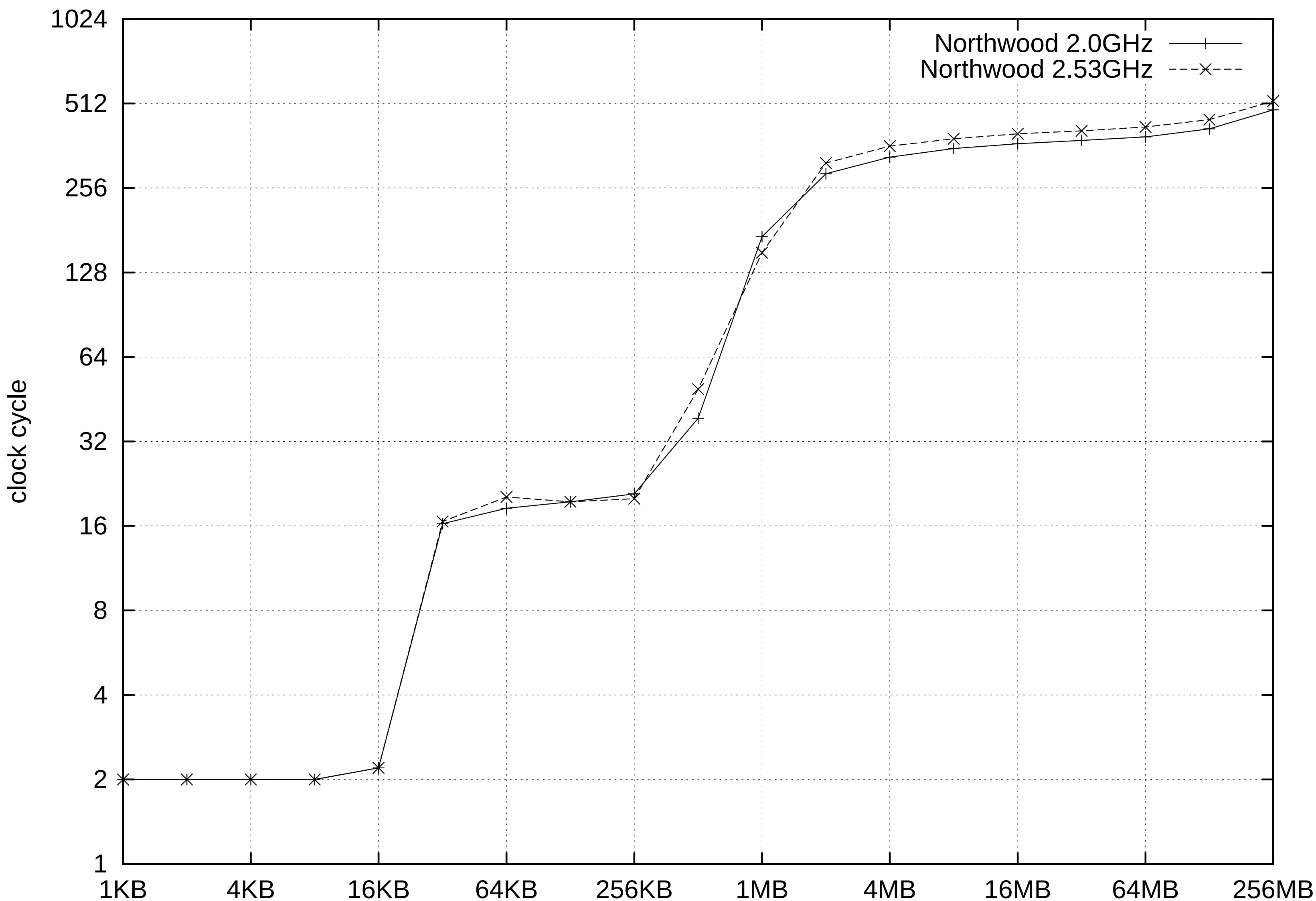
Memory Hierarchy of a PC

Pentium 4 Northwood

	Line size	Size	Latency
Register	(4 B)	32 B	$\frac{1}{2}$ pc
L1 cache	64 B	8 KB	2 pc
L2 cache	128 B	512 KB	7 pc
Main RAM	(4 KB)	$\approx$1 GB	12 pc $+$ 6-12 bc

pc: processor cycle, bc: bus cycle

Random Memory Read Latency of a PC



Number Field Sieve

- Developed at early 1990s
- The known fastest algorithm for general-type integer factoring
- Consists of many steps:
  - polynomial selection
  - sieving (most time-consuming part)
  - linear algebra
  - square root

Sieving

Find many (a, b) s.t.

$$|F(a,b)| = \prod_{p<B,\ p\text{: prime}} p^{e_p}, \quad (F \in \mathbf{Z}[x,y])$$

Use the following condition

$$p \mid F(a,b) \Rightarrow \begin{cases} p \mid F(a+p, b) \\ p \mid F(a, b+p) \end{cases}$$

line sieve

for $b \leftarrow 1$ **to** H_b
  initialize $\mathtt{S}[a]$ **to** $\log|F(a,b)|$ **(**$-H_a \leq \forall a < H_a$**)**
  for prime $p \leftarrow 2$ **to** B
    compute initial sieving point $a \geq -H_a$ **from** b **and** p
    while $a < H_a$
      $\mathtt{S}[a] \leftarrow \mathtt{S}[a] - \log p$
      $a \leftarrow a + p$
  completely factor $|F(a,b)|$ **if** $\mathtt{S}[a]$ **is small**

Memory Access while Sieving

- $(a, \log p)$**s are generated while sieving**
- **do "**$\mathrm{S}[a] \leftarrow \mathrm{S}[a] - \log p$**," using** $(a, \log p)$
- a**s are generated by step** p

Focusing on memory in PC,

the behaviors are very different as the size of p

$$\Downarrow$$

Classify primes according to their size.

Classification of Primes

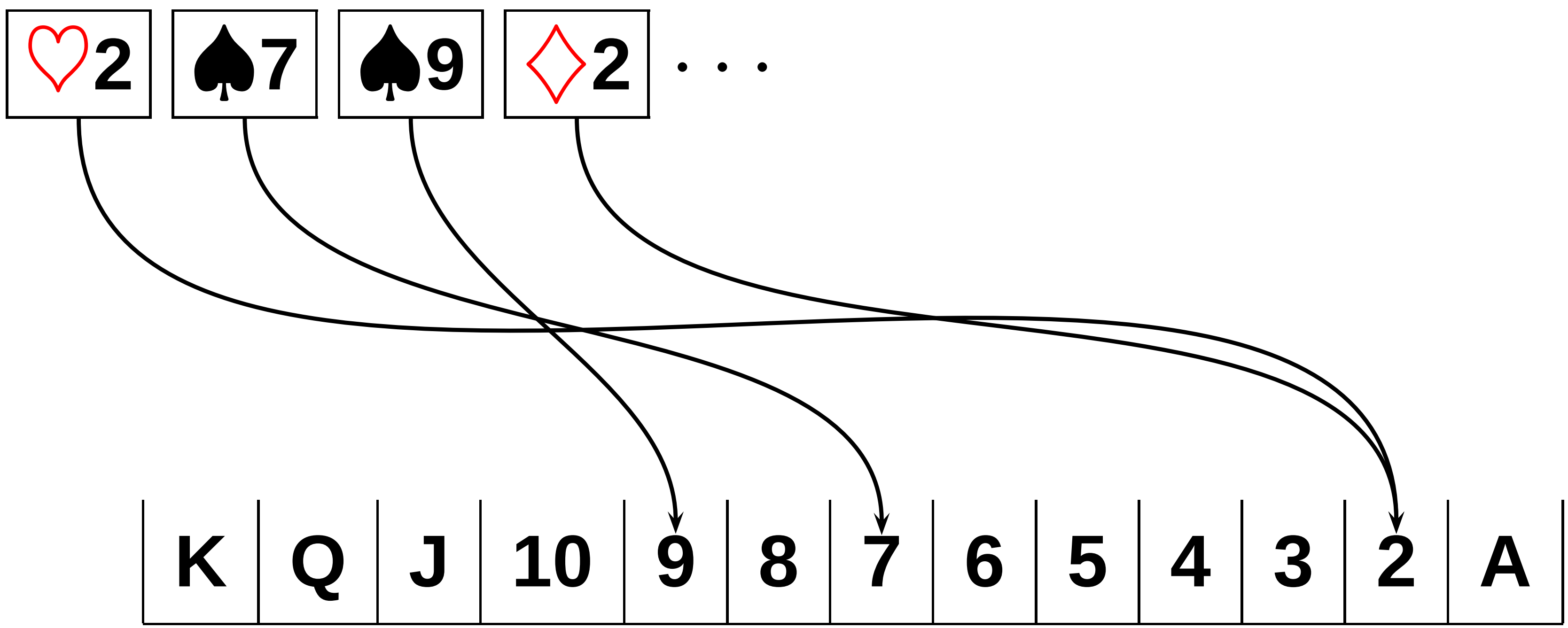
Range	Name	Algorithm
$p \leq B^T$	tiny prime	sieving pattern
$B^T < p \leq B^S$	smallish prime	block sieving
$B^S < p \leq B^L$	largish prime	bucket sort
$B^L < p \leq B$	large prime	primality testing and factoring

ex: $B^T = 5$, $B^S = 2^{19}$, $B^L = 2^{26}$, $B = 2^{32}$

Largish primes behave like random!

Bucket Sort

$\spadesuit\mathbf{K} \succ \heartsuit\mathbf{K} \succ \cdots \succ \diamondsuit\mathbf{A} \succ \clubsuit\mathbf{A}$



Bucket Filling Algorithm

1: make all buckets empty
2: for prime $p \leftarrow B^S + 1$ to B^L
3: compute initial sieving point $a \geq -H_a$
4: while $a < H_a$
5: Throw $(a, \log p)$ into $\left\lfloor \frac{a + H_a}{r} \right\rfloor$-th bucket
6: $a \leftarrow a + p$

original Step 5:

$$\mathtt{S}[a] \leftarrow \mathtt{S}[a] - \log p$$

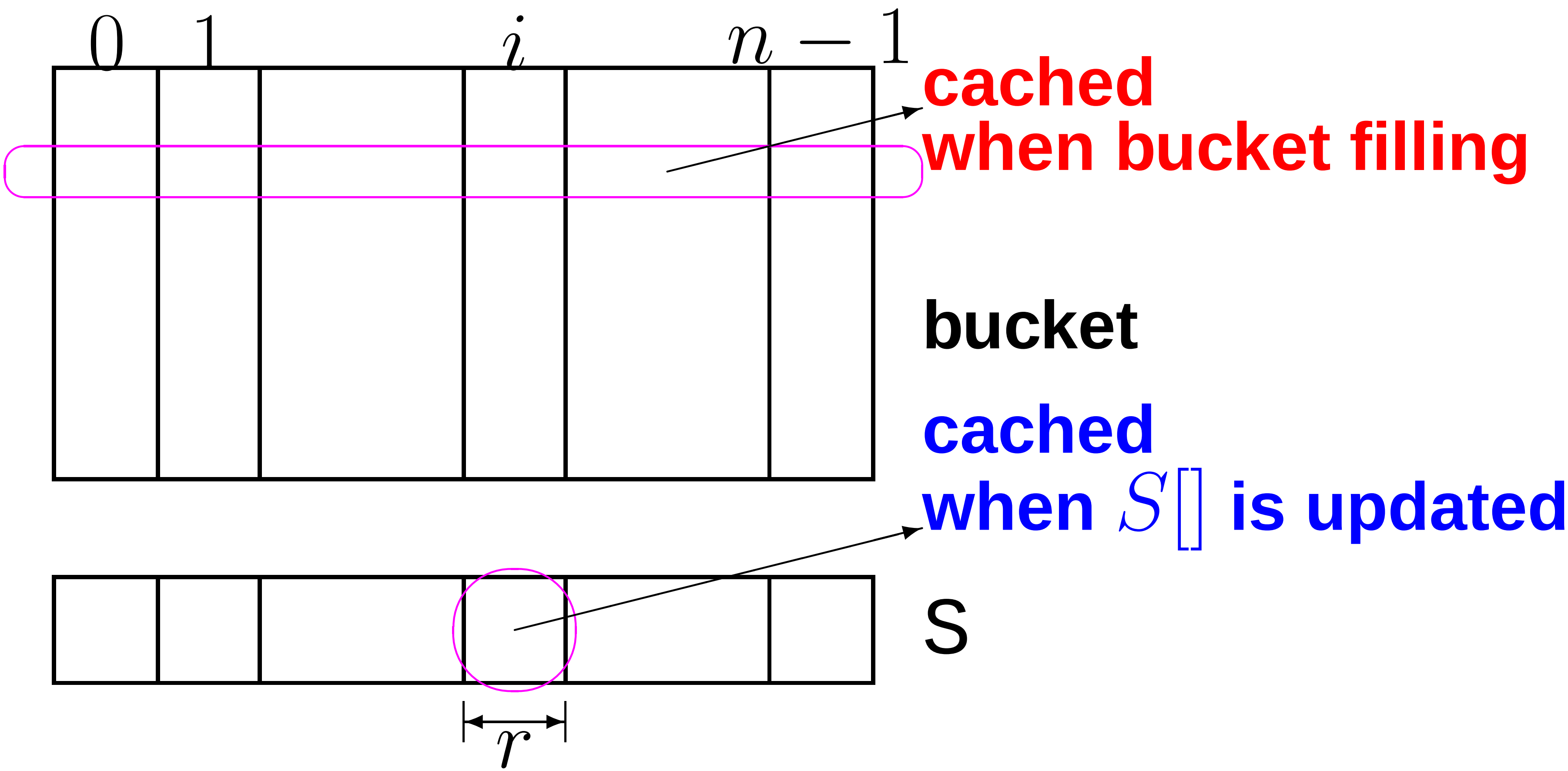
Continued writes are performed on each bucket.

$S[a]$ Update Algorithm

1: **for** i**-th bucket (**$0 \leq i < n$**)**
2: **for all** $(a, \log p)$ **in** i**-th bucket**
3: $\mathtt{S}[a] \leftarrow \mathtt{S}[a] - \log p$

- **Continued reads are performed on each bucket.**
- a **can only vary in the size of cache memory.**

Memory Map



0
1
i
n − 1
cached
when bucket filling
bucket
cached
when S[] is updated
S
r

Effective Condition

$$\text{(size of } \mathbb{S}[] \text{ area)} \leq \frac{\text{(size of cache)}^2}{\text{(size of cache line)}}$$

Example of Pentium 4 (Northwood):

L2 cache = 512KB,   cache line = 128B

$$\frac{\text{512KB}^2}{\text{128B}} = \text{2GB}$$

Franke's `gnfs-lasieve3e.tgz`

- **available at**
  `ftp://ftp.math.uni-bonn.de/people/franke/mpqs4linux/gnfs-lasieve3e.tgz`
- **time-stamped October 16, 2001.**
- **Focusing on L1 cache, and using the similar idea of bucket sort.**

Numerical Example for Lattice Sieve

Name of #	#bit	#LP	rel/MY
c164	545	2+2	29k
RSA-155	512	2+2	14k

Factoring Example

with Kida, Shimoyama, and Sonoda

Name of #	#bit	Method	Date
c164 in $2^{1826}+1$	545	GNFS	Dec 19, 2003
c248 in $2^{1642}+1$	822	SNFS	Apr 4, 2004

Both factorings spent about two month using 100 PCs.

Conclusion

Focusing on the memory hierarchy

+

Using the idea of bucket sort

$\Downarrow$

Several times faster sieving!