Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?

Daniel J. Bernstein *

Department of Computer Science (MC 152)
The University of Illinois at Chicago
Chicago, IL 60607–7053
djb@cr.yp.to

Abstract. Current proposals for special-purpose factorization hardware will become obsolete if large quantum computers are built: the number-field sieve scales much more poorly than Shor's quantum algorithm for factorization. Will *all* special-purpose cryptanalytic hardware become obsolete in a post-quantum world?

A quantum algorithm by Brassard, Høyer, and Tapp has frequently been claimed to reduce the cost of b-bit hash collisions from $2^{b/2}$ to $2^{b/3}$. This paper analyzes the Brassard–Høyer–Tapp algorithm and shows that it has fundamentally worse price-performance ratio than the classical van Oorschot–Wiener hash-collision circuits, even under optimistic assumptions regarding the speed of quantum computers.

Keywords. hash functions, collision-search algorithms, table lookups, parallelization, rho, post-quantum cryptanalysis

1 Introduction

The SHARCS (Special-Purpose Hardware for Attacking Cryptographic Systems) workshops have showcased a wide variety of hardware designs for factorization, brute-force search, and hash collisions. These hardware designs often achieve surprisingly good price-performance ratios and are among the top threats against currently deployed cryptosystems, such as RSA-1024.

Would any of this work be useful for a post-quantum attacker—an attacker equipped with a large quantum computer? The power of today's cryptanalytic hardware is of tremendous current interest, but will the

* Permanent ID of this document: 971550562a76ba87a7b2da14f71ca923. Date of this document: 2009.05.17. This work was supported by the National Science Foundation under grant ITR–0716498.

same hardware designs remain competitive in a world full of quantum computers, assuming that those computers are in fact built?

One might guess that the answer to both of these questions is no: that large quantum computers will become the tool of choice for all large cryptanalytic tasks. Should SHARCS prepare for a transition to a post-quantum SHARCS?

Case study: Factorization. Today's public efforts to build factorization hardware are focused on the number-field sieve. The number-field sieve is conjectured to factor b-bit RSA moduli in time $2^{b^{1/3+o(1)}}$; older algorithms take time $2^{b^{1/2+o(1)}}$ and do not appear to be competitive with the number-field sieve once b is sufficiently large. Detailed analyses show that "sufficiently large" includes a wide range of b's of real-world cryptographic interest, notably $b = 1024$.

The standard advertising for quantum computers is that they can factor much more efficiently than the number-field sieve. Specifically, Shor in [15] and [16] introduced an algorithm to factor a b-bit integer in $b^{\Theta(1)}$ operations on a quantum computer having $b^{\Theta(1)}$ qubits. For a detailed analysis of the number of qubit operations inherent in Shor's algorithm see, e.g., [19].

Simulating this quantum computer on traditional hardware would make it exponentially slower. The goal of quantum-computer engineering is to directly build qubits as physical devices that can efficiently and reliably carry out quantum operations. Note that, thanks to "quantum error correction," perfect reliability is not required; for example, [4, Section 5.3.3.3] shows that an essentially perfect qubit can be simulated by an essentially constant number of 99.99%-reliable qubits.

Assume that this goal is achieved, and that a quantum computer can be built for $b^{\Theta(1)}$ Euros to factor a b-bit integer in $b^{\Theta(1)}$ seconds. This quantum computer will be much more scalable than number-field-sieve hardware, and therefore much more cost-effective than number-field-sieve hardware for large b—including b's of cryptographic interest if the exponents $\Theta(1)$ are reasonably small.

Case study: Preimage search. Similar comments apply to hardware for brute-force-search hardware, i.e., hardware to compute preimages.

Consider a function H that can be computed by a straight-line sequence of h bit operations. Assume for simplicity that there is a unique b-bit string x satisfying $H(x) = 0$. Grover in [8] and [9] presented a quantum algorithm to find this x with high probability in approximately $2^{b/2}h$ operations on $\Theta(h)$ qubits. A real-world quantum computer with

similar performance would scale much more effectively than traditional hardware using $2^b h$ operations, and would therefore be much more cost-effective than traditional hardware for large b—again possibly including b's of cryptographic interest. Grover's speedup from $2^b h$ to $2^{b/2} h$ is not as dramatic as Shor's speedup from $2^{b^{1/3+o(1)}}$ to $b^{\Theta(1)}$, but it is still a clear speedup when b is large.

More generally, assume that there are exactly p preimages of 0 under H. Traditional hardware finds a preimage with high probability using approximately $(2^b/p)h$ operations. Boyer, Brassard, Høyer, and Tapp in [5] presented a minor extension of Grover's algorithm to find a preimage with high probability using approximately $(2^{b/2}/p^{1/2})h$ quantum operations on $\Theta(h)$ qubits. It is not necessary for p to be known in advance.

If quantum search is run for only $\epsilon(2^{b/2}/p^{1/2})h$ operations then it has approximately an ϵ^2 chance of success.

Case study: Collision search. The point of this paper is that all known quantum algorithms to find collisions in hash functions are *less* cost-effective than traditional cryptanalytic hardware, even under optimistic assumptions regarding the speed of quantum computers. Quantum computers win for sufficiently large factorizations, and for sufficiently large preimage searches, but they do not win for collision searches.

This conclusion does not depend on the engineering difficulty of building quantum computers; it will remain true even in a world full of quantum computers. This conclusion also does not depend on real-world limits on interesting input sizes. Within the space of known quantum collision algorithms, the most cost-effective algorithms are tantamount to non-quantum algorithms, and it is clear that non-quantum algorithms should be implemented with standard bits rather than with qubits.

In particular, this paper shows that the quantum collision method introduced in [6] by Brassard, Høyer, and Tapp is fundamentally less cost-effective than the collision-search circuits that had been introduced years earlier by van Oorschot and Wiener in [17]. There is a popular myth that the Brassard–Høyer–Tapp algorithm reduces the cost of b-bit hash collisions from $2^{b/2}$ to $2^{b/3}$; this myth rests on a nonsensical notion of cost and is debunked in this paper.

Figures 1.1 and 1.2 summarize the asymptotic speeds of the attack machines considered in this paper. The horizontal axis is machine size, from 2^0 to $2^{b/2}$. The vertical axis is (typical) time to find a collision, from 2^0 to 2^b. Figure 1.1 assumes a realistic two-dimensional communication mesh; Figure 1.2 makes the naive assumption that communication is free.

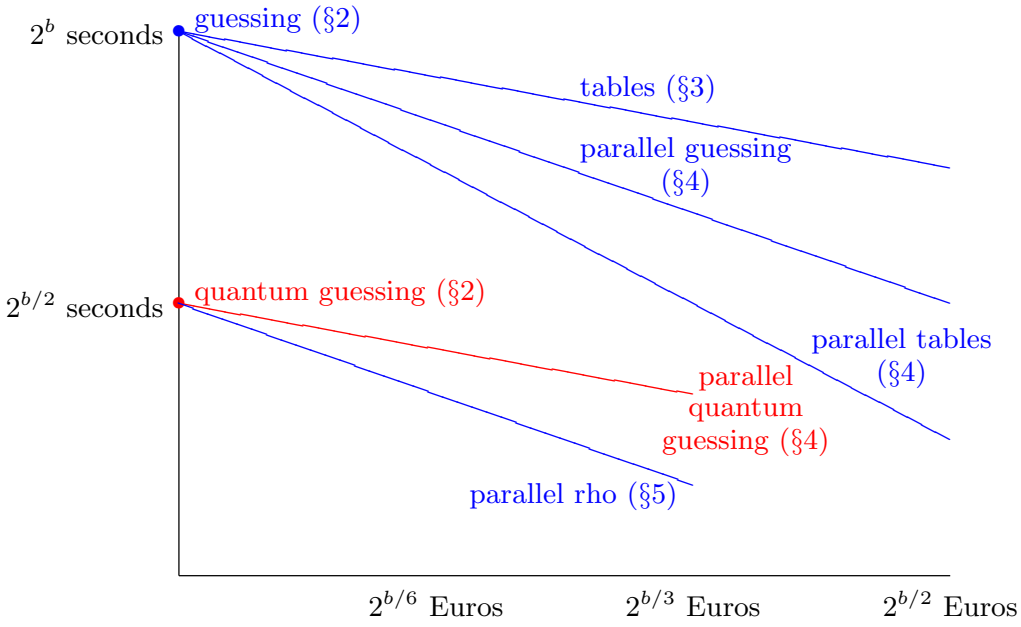

Fig. 1.1. Asymptotic collision-search time assuming realistic communication costs. Parallel rho is 1994 van Oorschot–Wiener [17]. Parallel quantum guessing is 2003 Grover–Rudolph [10].

2 Guessing a collision

A collision in a function H is, by definition, a pair (x, y) such that $x \neq y$ and $H(x) = H(y)$. The simplest way to find a collision is to simply guess a pair (x, y) in the domain of H and see whether it is a collision.

Assume, for concreteness, that H maps $(b + c)$-bit strings to b-bit strings, where $c \geq 1$. Assume that x and y are uniform random $(b+c)$-bit strings. What is the chance that (x, y) is a collision in H? The answer depends on the distribution of output values of H but is guaranteed to be at least $1/2^b - 1/2^{b+c}$. The proof is a standard calculation: say the 2^b output values of H have $p_0, p_1, \ldots, p_{2^b-1}$ preimages respectively, where $p_0 + p_1 + \cdots + p_{2^b-1} = 2^{b+c}$; then the number of collisions (x, y) is

$$\begin{aligned}
&p_0(p_0 - 1) + p_1(p_1 - 1) + \cdots + p_{2^b-1}(p_{2^b-1} - 1) \\
&\quad = p_0^2 + p_1^2 + \cdots + p_{2^b-1}^2 - (p_0 + p_1 + \cdots + p_{2^b-1}) \\
&\quad \geq \frac{(p_0 + p_1 + \cdots + p_{2^b-1})^2}{2^b} - (p_0 + p_1 + \cdots + p_{2^b-1}) \\
&\quad = 2^{b+2c} - 2^{b+c}
\end{aligned}$$

by Cauchy's inequality.

A sequence of N independent guesses succeeds with probability at least $1 - (1 - (1/2^b - 1/2^{b+c}))^N$ and involves at worst $2N$ computations

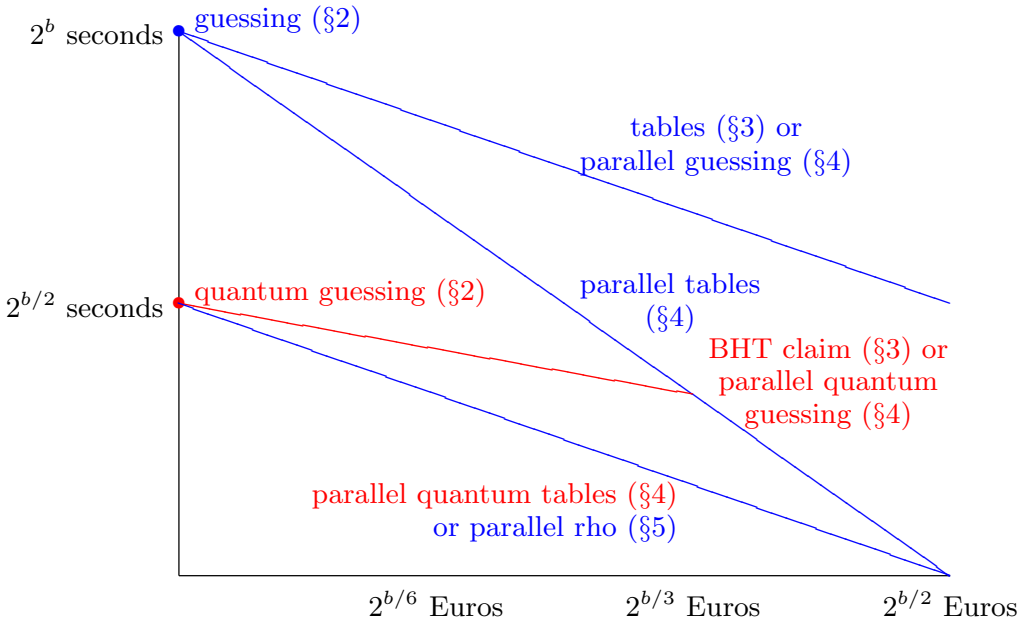

Fig. 1.2. Asymptotic collision-search time assuming free communication. Parallel rho is 1994 van Oorschot–Wiener [17]. "BHT claim" is 1998 Brassard–Høyer–Tapp [6]. Parallel quantum guessing is 2003 Grover–Rudolph [10].

of H. In particular, a sequence of $\lceil 1/(1/2^b - 1/2^{b+c}) \rceil \approx 2^b$ independent guesses succeeds with probability more than $1 - \exp(-1) \approx 0.63$ and involves (at worst) $\approx 2^{b+1}$ computations of H. This attack can be implemented on a very small circuit, typically dominated by the size of a circuit to compute H.

The impact of quantum computers. A collision in H is a preimage of 0 under the $(2b + 2c)$-bit-to-b-bit function $(x, y) \mapsto H(x) \oplus H(y)$. One can find a preimage by quantum search instead of by guessing. Quantum search uses approximately $2^{b/2}h$ quantum operations on $\Theta(h)$ qubits, where h is the cost of evaluating the function.

One could summarize this change by claiming that quantum computers reduce collision-search time from 2^b to $2^{b/2}$, saving a factor of $2^{b/2}$. There are two reasons that the actual speedup factor is much smaller. The first reason is that, even in the most optimistic visions of quantum computing, qubits will be larger and slower than bits. The second reason is that there are many other ways to reduce the time far below 2^b, and in fact far below $2^{b/2}$, without quantum computing. There are also faster quantum collision-search algorithms, but—as shown in subsequent sections of this paper—the non-quantum algorithms are the most cost-effective algorithms known.

3 Table lookups

There is a classic way to use large tables to reduce the number of H evaluations:

- Generate many inputs $x_1, x_2, \ldots, x_M$.
- Compute $H(x_1), H(x_2), \ldots, H(x_M)$, and lexicographically sort the M pairs $(H(x_1), x_1), (H(x_2), x_2), \ldots, (H(x_M), x_M)$.
- Generate many more inputs $y_1, y_2, \ldots, y_N$. After generating y_j, compute $H(y_j)$ and look it up in the sorted list, hoping to find a collision.

This attack has the same effect as searching all MN pairs (x_i, y_j) for collisions $H(x_i) = H(y_j)$. In particular, the attack has a high probability of success if $M \approx N \approx 2^{b/2}$. What makes the attack interesting is that it is faster than considering each pair (x_i, y_j) separately—although it also requires a large attack machine with $M(2b + c)$ bits of memory.

In a naive model of communication, random access to a huge array takes constant time; looking up $H(y_i)$ in the sorted list takes approximately $\lg M$ memory accesses; and sorting in the first place takes approximately $M \lg M$ memory accesses. The table-lookup attack thus takes $M+N$ evaluations of H and additional time approximately $(M+N) \lg M$ for memory access. For example, if $M \approx N \approx 2^{b/2}$, then the attack takes $2^{b/2+1}$ evaluations of H and additional time approximately $2^{b/2}b$ for memory access.

In a realistic two-dimensional model of communication, random access to an N-element array takes time $N^{1/2}$. The table-lookup attack thus takes $M + N$ evaluations of H and additional time approximately $(M + N)M^{1/2} \lg M$ for memory access. For example, if $M \approx N \approx 2^{b/2}$, then the attack takes $2^{b/2+1}$ evaluations of H and additional time approximately $2^{3b/4}b$ for memory access. Memory access is the dominant cost here for typical choices of H.

To summarize, a size-M machine finds collisions in time roughly $2^b/M$ in a naive model of communication, or time roughly $2^b/M^{1/2}$ in a realistic model of communication. If this machine is run for only ϵ times as long then it has approximately an ϵ chance of success.

The impact of quantum computers. Fix $x_1, x_2, \ldots, x_M$. Consider the b-bit-to-1-bit function F defined as follows: $F(y) = 0$ if there is a collision among $(x_1, y), (x_2, y), \ldots, (x_M, y)$; otherwise $F(y) = 1$. The above attack guesses a preimage of 0 under F.

Brassard, Høyer, and Tapp in [6] propose instead finding a preimage of F by quantum search. They claim in [6, Section 3] that this quantum

attack takes "expected time $O((k + \sqrt{N/rk})(T + \log k))$" where "$N$" is the number of hash-function inputs (i.e., 2^{b+c}), "N/r" is the number of hash-function outputs (i.e., 2^b), "k" is the table size (i.e., M), and "T" is the cost of evaluating the hash function (i.e., h). In other words, they state that quantum search finds a preimage of F in expected time $O((M + 2^{b/2}/M^{1/2})(h + \log M))$.

There are several reasons to question the Brassard–Høyer–Tapp claim. Quantum search uses $2^{b/2}/M^{1/2}$ evaluations of F, not merely $2^{b/2}/M^{1/2}$ evaluations of H. Computing $F(y)$ requires not only computing $H(y)$ but also comparing $H(y)$ to $H(x_1), H(x_2), \ldots, H(x_M)$. There are two obstacles to performing these comparisons efficiently when M is large:

- Realistic two-dimensional models of quantum computation, just like realistic models of non-quantum computation, need time $M^{1/2}$ for random access to a table of size M. This $M^{1/2}$ loss is as large as the $M^{1/2}$ speedup claimed by Brassard, Høyer, and Tapp.
- A straight-line circuit to compare $H(y)$ to $H(x_1), H(x_2), \ldots, H(x_M)$ uses $\Theta(Mb)$ bit operations, so a quantum circuit has to use $\Theta(Mb)$ qubit operations. Sorting the table $H(x_1), H(x_2), \ldots, H(x_M)$ does not reduce the size of a *straight-line* comparison circuit, so it does not reduce the number of quantum operations. The underlying problem is that, inside the quantum search, the input to the comparison is a quantum superposition of b-bit strings, so the output depends on all Mb bits in the precomputed table.

There are much simpler quantum collision-search algorithms that reach the speed that Brassard, Høyer, and Tapp claim for their algorithm; see the next section of this paper. Unfortunately, as discussed later, this speed is still not competitive with non-quantum collision hardware.

4 Parallelization

There is a much simpler way to build a machine of size M that finds collisions in time $\Theta((2^b/M)h)$. The machine consists of M small independent collision-guessing units, all running in parallel. This machine does as much work in time T as a single collision-guessing machine would do in time MT. In particular, it has high probability of finding a collision in time $\Theta((2^b/M)h)$—not just in a naive model of communication, but in a realistic model of communication. This machine, unlike the table-lookup machine described in the previous section, does not have trouble with communication as M grows.

A more sophisticated size-M machine sorts $H(x_1), H(x_2), \ldots, H(x_M)$ and $H(y_1), H(y_2), \ldots, H(y_M)$ in time $\Theta(bM^{1/2})$ using a two-dimensional mesh-sorting algorithm; see, e.g., [14] and [13]. Computing the H values takes time $\Theta(h)$; the sorting dominates if M is large. This machine has probability approximately $M^2/2^b$ of finding a collision, assuming $M \leq 2^{b/2}$. Repeating the same procedure $2^b/M^2$ times takes time only $\Theta(2^b b/M^{3/2})$ and has high probability of finding a collision.

To summarize, this size-M machine finds collisions in time roughly $2^b/M^{3/2}$ in a realistic model of communication. For example, a machine of size $2^{b/3}$ finds collisions in time roughly $2^{b/2}$. If this machine is run for ϵ times as long then it has approximately an ϵ chance of success.

The impact of quantum computers. Consider a size-M quantum computer that consists of M small independent collision-searching units, all running in parallel. After approximately $2^{b/2}h\epsilon$ quantum operations, each collision-searching unit has approximately an ϵ^2 chance of success, so the entire machine has approximately an $M\epsilon^2$ chance of success. In particular, the machine has a high probability of success after approximately $2^{b/2}h/M^{1/2}$ quantum operations.

For example, a quantum computer of size $2^{b/3}$ can find collisions in time approximately $2^{b/3}$, as claimed in [6]. The fact that mindless parallelism would achieve the same performance as [6] was pointed out by Grover and Rudolph in [10].

One can also try to build the quantum analogue of the more sophisticated size-M machine discussed above. Consider the function F that, given $2Mb$ bits $(x_1, x_2, \ldots, x_M, y_1, y_2, \ldots, y_M)$, outputs 0 if and only if some (x_i, y_j) is a collision in H. Quantum search finds a preimage of F using approximately $2^{b/2}/M$ quantum evaluations of F, saving a factor of $M^{1/2}$ compared to the previous algorithm. A standard two-dimensional mesh-sorting algorithm to compute F can be converted into a two-dimensional mesh-sorting quantum algorithm taking time $\Theta(bM^{1/2})$ on a machine of size M.

This more sophisticated machine might be slightly better than the mindlessly parallel quantum machine if H is expensive, but its overall time is still on the scale of $2^{b/2}/M^{1/2}$. The benefit of considering M inputs together is that M operations produce M^2 collision opportunities, a factor M better than mindless parallelism—but this speeds up quantum search by only $M^{1/2}$, while communication costs also grow by a factor $M^{1/2}$.

The same idea would be an improvement over [6] and [10] in a three-dimensional model of parallel quantum computation, or in a naive parallel model without communication delays. The function F can be evaluated

by a straight-line sequence of $\Theta(bM)$ bit operations (using radix sort), and if communication were free then a size-M machine could carry out all of those bit operations in essentially constant time. For example, a quantum computer of size $2^{b/3}$ would be able to find collisions in time approximately $2^{b/6}$ in a naive model.

5 The rho method

Let me review. The best size-M non-quantum machine described so far takes time roughly $2^b/M^{3/2}$ to find a collision: for example, $2^{b/2}$ if $M = 2^{b/3}$, or $2^{b/4}$ in the extreme case $M = 2^{b/2}$. Quantum search reduces $2^b/M^{3/2}$ to $2^{b/2}/M^{1/2}$: for example, $2^{b/3}$ if $M = 2^{b/3}$, or $2^{b/4}$ in the extreme case $M = 2^{b/2}$. All of these results are for a realistic model of communication. A naive model reduces $2^{b/2}, 2^{b/4}, 2^{b/3}, 2^{b/4}$ to $2^{b/3}, 1, 2^{b/6}, 1$.

"But what about the rho method?" the cryptographers are screaming. "What kind of idiot would use a machine of size $2^{b/3}$ to find collisions in time $2^{b/2}$, when everybody knows how to use a tiny machine to find collisions just as quickly?"

Recall that the rho method *iterates* the function H. Choose a $(b+c)$-bit string x_0, compute the b-bit string $H(x_0)$, apply an injective padding function π to produce a $(b+c)$-bit string $x_1 = \pi(H(x_0))$, compute $H(x_1)$, compute $x_2 = \pi(H(x_1))$, etc. After approximately $2^{b/2}$ steps one can reasonably expect to find a "distinguished point": a string x_i whose first $b/2$ bits are all 0. (In practice very simple functions π such as "append c zero bits" seem to work for every function H of cryptographic interest, although theorems obviously require more randomness in π.)

Now consider another such sequence $y_0, y_1, \ldots$, again iterated until a distinguished point. There are approximately 2^b pairs (x_i, y_j) before those distinguished points, so one can reasonably expect that those pairs include a collision. Furthermore, if those pairs *do* include a collision, then the distinguished points will be identical; the sequence lengths will then reveal the difference $i - j$, and an easy recomputation of the sequences will find the collision.

More generally, consider a machine with M parallel iterating units, and redefine a "distinguished point" as a string x_i whose first $b/2 - \lceil \lg M \rceil$ bits are all 0. In time approximately $2^{b/2}/M$ this machine will have considered $\Theta(2^{b/2})$ inputs to H and will have found $\Theta(M)$ distinguished points. The inputs have a good chance of including a collision, and that collision is easily found from a match in the distinguished points. Sorting the dis-

tinguished points takes time only $\Theta(M^{1/2})$; this is not a bottleneck for $M \leq 2^{b/3}$.

To summarize, this size-M machine finds collisions in time roughly $2^{b/2}/M$. For example, a machine of size 1 finds collisions in time roughly $2^{b/2}$; a machine of size $2^{b/6}$ finds collisions in time roughly $2^{b/3}$; and a machine of size $2^{b/3}$ finds collisions in time roughly $2^{b/6}$. All of these results hold in a realistic model of communication.

The special case $M = 1$ was introduced by Pollard in [12] in 1975. The general case, finding collisions in time $2^{b/2}/M$, was introduced by van Oorschot and Wiener in [17] in 1994.

The impact of quantum computers. All of the quantum-collision algorithms in the literature are steps backwards from the non-quantum algorithm of [17].

The best time claimed—by Brassard, Høyer, and Tapp in [6], and by Grover and Rudolph in [10]—is $2^{b/2}/M^{1/2}$ on a size-M quantum computer. This is no better than running M parallel copies of Pollard's 1975 method, and is much worse than the van Oorschot–Wiener method.

The previous section of this paper explains how to achieve time $2^{b/2}/M$ on a size-M quantum computer, but only in a naive model allowing free communication. The design has to evaluate H as many times as the van Oorschot–Wiener method, and has to evaluate it on qubits rather than on bits. The lack of iteration in this design might be pleasing for purists who insist on proofs of performance, but this feature is of no practical interest.

Of course, one can also achieve quantum time $2^{b/2}/M$ by viewing the van Oorschot–Wiener algorithm as a quantum algorithm. However, replacing bits with qubits certainly does not save time! There are several ways to combine quantum search with the rho method, but there is no evidence that any such combinations *improve* performance; quantum search allows N operations to search N^2 possibilities, but the rho method already has the same efficiency.

Many authors have claimed that quantum computers will have an impact on the complexity of hash collisions, reducing time $2^{b/2}$ to time $2^{b/3}$. In fact, time $2^{b/3}$ had already been achieved by non-quantum machines of size just $2^{b/6}$, and smaller time $2^{b/4}$ had already been achieved by non-quantum machines of size $2^{b/4}$. Anyone afraid of quantum hash-collision algorithms already has much more to fear from non-quantum hash-collision algorithms.

References

1. — (no editor), *Proceedings of the 18th annual ACM symposium on theory of computing*, Association for Computing Machinery, New York, 1986. ISBN 0–89791–193–8. See [14].
2. — (no editor), *2nd ACM conference on computer and communication security, Fairfax, Virginia, November 1994*, Association for Computing Machinery, 1994. See [17].
3. — (no editor), *Proceedings of the twenty-eighth annual ACM symposium on the theory of computing, held in Philadelphia, PA, May 22–24, 1996*, Association for Computing Machinery, 1996. ISBN 0-89791-785-5. MR 97g:68005. See [8].
4. Panos Aliferis, *Level reduction and the quantum threshold theorem* (2007). URL: `http://arxiv.org/abs/quant-ph/0703230`. Citations in this document: §1.
5. Michel Boyer, Gilles Brassard, Peter Høyer, Alain Tapp, *Tight bounds on quantum searching* (1996). URL: `http://arxiv.org/abs/quant-ph/9605034v1`. Citations in this document: §1.
6. Gilles Brassard, Peter Høyer, Alain Tapp, *Quantum cryptanalysis of hash and claw-free functions*, in [11] (1998), 163–169. MR 99g:94013. Citations in this document: §1, §1.2, §1.2, §3, §3, §4, §4, §4, §5.
7. Shafi Goldwasser (editor), *35th annual IEEE symposium on the foundations of computer science. Proceedings of the IEEE symposium held in Santa Fe, NM, November 20–22, 1994*, IEEE, 1994. ISBN 0-8186-6580-7. MR 98h:68008. See [15].
8. Lov K. Grover, *A fast quantum mechanical algorithm for database search*, in [3] (1996), 212–219. MR 1427516. Citations in this document: §1.
9. Lov K. Grover, *Quantum mechanics helps in searching for a needle in a haystack*, Physical Review Letters **79** (1997), 325–328. Citations in this document: §1.
10. Lov K. Grover, Terry Rudolph, *How significant are the known collision and element distinctness quantum algorithms?*, Quantum Information & Computation **4** (2003), 201–206. MR 2005c:81037. URL: `http://arxiv.org/abs/quant-ph/0309123`. Citations in this document: §1.1, §1.1, §1.2, §1.2, §4, §4, §5.
11. Claudio L. Lucchesi, Arnaldo V. Moura (editors), *LATIN'98: theoretical informatics. Proceedings of the 3rd Latin American symposium held in Campinas, April 20–24, 1998*, Lecture Notes in Computer Science, 1380, Springer, 1998. ISBN ISBN 3-540-64275-7. MR 99d:68007. See [6].
12. John M. Pollard, *A Monte Carlo method for factorization*, BIT **15** (1975), 331–334. ISSN 0006–3835. MR 52:13611. URL: `http://cr.yp.to/bib/entries.html#1975/pollard`. Citations in this document: §5.
13. Manfred Schimmler, *Fast sorting on the instruction systolic array*, report 8709, Christian-Albrechts-Universität Kiel, 1987. Citations in this document: §4.
14. Claus P. Schnorr, Adi Shamir, *An optimal sorting algorithm for mesh-connected computers*, in [1] (1986), 255–261. Citations in this document: §4.
15. Peter W. Shor, *Algorithms for quantum computation: discrete logarithms and factoring.*, in [7] (1994), 124–134; see also newer version [16]. MR 1489242. Citations in this document: §1.
16. Peter W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal on Computing **26** (1997), 1484–1509; see also older version [15]. MR MR 98i:11108. Citations in this document: §1.
17. Paul C. van Oorschot, Michael Wiener, *Parallel collision search with application to hash functions and discrete logarithms*, in [2] (1994), 210–218; see also newer version [18]. Citations in this document: §1, §1.1, §1.1, §1.2, §1.2, §5, §5.

18. Paul C. van Oorschot, Michael Wiener, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12** (1999), 1–28; see also older version [17]. ISSN 0933–2790. URL: `http://members.rogers.com/paulv/papers/pubs.html`.
19. Christof Zalka, *Fast versions of Shor's quantum factoring algorithm* (1998). URL: `http://arxiv.org/abs/quant-ph/9806084`. Citations in this document: §1.