Are Your Passwords Safe: Energy-Efficient Bcrypt Cracking with Low-Cost Parallel Hardware

Katja Malvoni
katja.malvoni@fer.hr
University of Zagreb, Croatia

Solar Designer
solar@openwall.com
Openwall

Josip Knezovic
josip.knezovic@fer.hr
University of Zagreb, Croatia

Abstract

Bcrypt is a password hashing scheme based on the Blowfish block cipher. It was designed to be resistant to brute force attacks and to remain secure despite of hardware improvements [13]. Expensive key setup with user-defined cost setting makes this hash slow while rapid random 32-bit lookups using Blowfish's variable S-boxes require 4 KB of local memory per instance. This memory access pattern makes bcrypt moderately unfriendly to parallel implementation on modern CPUs, where on one hand gather addressing is required in order to exploit the CPUs' SIMD capabilities, and on the other even when gather addressing is in fact available the L1 data cache size becomes the limiting factor. Despite of this (and due to it), it is possible to achieve much better performance per Watt with bcrypt implementations on homogeneous and heterogeneous multiprocessing platforms: Parallella board with 16- or 64-core Epiphany accelerator and ZedBoard with Zynq reconfigurable logic [16, 2]. Proposed implementations were integrated into John the Ripper password cracker resulting in improved energy efficiency by a factor of 35+ compared to heavily optimized implementations on modern CPUs.

1 Introduction

Password hashing was introduced in the 1970s to avoid storing passwords in plaintext. Instead, passwords are hashed and only the hashes are stored, which prevents attackers from directly obtaining the actual passwords. However, in many cases passwords may nevertheless be inferred by probing likely (and even not so likely) candidate passwords against the hashes. To mitigate these attacks, specialized password hashing schemes were designed, and bcrypt is one of them [13, 4]. It was designed to remain secure despite hardware improvements and to be resistant to brute-force attacks. The original goal is mostly achieved (so far) as it relates to bcrypt hash cracking on general-purpose CPUs and even on GPUs. On general-purpose CPUs, cracking-optimized implementations achieve a speedup of roughly a factor of 2 compared to optimal defensive implementations running on the same CPU cores, and GPUs achieve CPU-like attack performance only (although this might be changing). This work shows that certain other hardware platforms available today make it possible to optimize bcrypt hash cracking much further, especially in terms of energy efficiency. Although optimizing for specific platforms reduces portability, it does not affect scalability within the same hardware platform family.

The rest of the paper is organized as follows. Section 2 gives background details needed to understand energy-efficient bcrypt implementations. Section 3 gives an overview of various bcrypt implementations including underlying hardware architecture details and architecture-specific optimizations. In Section 4, energy-efficient bcrypt implementations are compared to existing CPU and GPU implementations in terms of performance and energy efficiency. Section 5 presents our equation for estimating theoretical bcrypt peak performance on a specific platform. Related work is discussed in Section 6. Finally, Section 7 contains the conclusion.

2 Background

Bcrypt is a password hashing scheme based on the Blowfish block cipher, which is structured as a 16-round Feistel network [13]. Blowfish encryption uses a 64-bit input and a P-box (in bcrypt, it initially holds the password being hashed) to calculate addresses used to access four 1 KB large S-boxes. These memory accesses are pseudorandom and 32-bit wide. Blowfish encryption is used by the EksBlowfish algorithm (Algorithm 1) in function ExpandKey() to derive state determined by values stored in S-boxes and P-box. This algorithm has three inputs: cost, salt and encryption key (password being hashed). Cost determines how expensive the key

setup process is, salt is a 128-bit random value used so that the same password does not always have the same hash value and encryption key is the user chosen password (after trivial pre-processing) [13]. Although this is how EksBlowfish is defined in [13], the order of lines 4 and 5 in Algorithm 1 is swapped in actual implementations, including in OpenBSD's original that pre-dates the USENIX 1999 paper by two years.

Algorithm 1 EksBlowfishSetup(cost, salt, key) [13]

1: $state \leftarrow InitState()$
2: $state \leftarrow ExpandKey(state, salt, key)$
3: $repeat(2^{cost})$
4: $\quad state \leftarrow ExpandKey(state, 0, salt)$
5: $\quad state \leftarrow ExpandKey(state, 0, key)$
6: $return\ state$

Bcrypt (Algorithm 2) runs in two phases. The first phase uses the expensive key schedule Blowfish algorithm (Algorithm 1) to initialize Blowfish state. In the second phase, the obtained state is used with Blowfish in electronic codebook (ECB) mode to encrypt the 192-bit string "OrpheanBeholderScryDoubt" 64 times. Returned value is bcrypt hash [13].

Algorithm 2 bcrypt(cost, salt, pwd) [13]

1: $state \leftarrow EksBlowfishSetup(cost, salt, key)$
2: $ctext \leftarrow$ "$OrpheanBeholderScryDoubt$"
3: $repeat(64)$
4: $\quad ctext \leftarrow EncryptECB(state, ctext)$
5: $return\ Concatenate(cost, salt, ctext)$

3 Energy-Efficient Implementations

This section gives details of bcrypt implementation on different energy-efficient platforms: Parallella board with 16- or 64-core Epiphany manycore accelerator [2, 3] and ZedBoard with Zynq 7020 reconfigurable logic [16, 17]. In these implementations, most resources are spent to optimize the most time consuming part of bcrypt, which is loop executed 2^{cost} times (Algorithm 1, Lines 3 to 5).

3.1 Parallella/Epiphany

Parallella board consists of ARM Cortex-A9 CPU, 16 or 64-core Epiphany manycore accelerator and Zynq reconfigurable logic. Each Epiphany core has 32 KB of local memory, 64 registers, integer ALU unit with single cycle latency and floating point unit (FPU), which can also be used in integer mode [1]. Limitations of the described architecture include missing support for complex addressing modes and incapability of floating point unit to issue logic instructions. One cycle instruction latency (including load instructions from local memory) and 32 KB of local memory overcome bcrypt's random memory access pattern while floating point unit used in integer mode can further exploit some of available instruction level parallelism. Atypically high number of registers allows to preload the P-box, which further improves performance. Performance improvement is constrained by missing support for complex addressing modes (causes register waste during S-box lookups) and floating point unit without support for logic instructions (prevents additional exploitation of instruction level parallelism). However, advantages of this architecture surpass limitations to a great extent, which results in performance efficient and energy-efficient bcrypt implementations. To exploit all advantages of underlying hardware architecture, bcrypt was implemented on Epiphany manycore accelerator where each core computes bcrypt hashes while ARM CPU is used by John the Ripper [9] to generate candidate passwords and send them to Epiphany cores for hash computations. In order to hide the four cycle latency of FPU configured in integer mode and FPU's inability to issue logic instructions, it was necessary to introduce more instruction level parallelism. Computation of a single bcrypt hash per core could not exploit available resources. Therefore, we overlapped two bcrypt computations on each core. Instruction level parallelism was further exploited by partial interleaving computations of two Blowfish encryption rounds for each instance, which sums up to four Blowfish encryption rounds being interleaved. These optimizations allow some instructions to be calculated for free, i.e. one instruction is executed on ALU while another one is executed on FPU. Apart from additional instruction level parallelism, P-box for both instances was preloaded in 36 registers to avoid additional load instructions. Described optimizations were implemented with portions of code in assembly, namely for lines 3 to 5 of Algorithm 1. With this approach we managed to achieve 3/4th of the per-MHz per-core speed of a full integer dual-issue architecture.

3.2 ZedBoard/Zynq 7020

ZedBoard is a heterogeneous platform, which consists of Zynq–7020 all-programmable device featuring dual ARM Cortex-A9 CPU and reconfigurable logic [16, 17]. Given the heterogeneous nature of the Zynq–7020 device, we implemented time consuming parts of bcrypt in reconfigurable logic while leaving less demanding tasks on ARM cores. Figure 1 illustrates our approach to partitioning the algorithm: candidate passwords are computed on ARM CPU and sent to arbiter module using

AXI4 bus. Arbiter receives data and stores it in corresponding block RAMs. After receiving data, arbiter starts bcrypt instances running in parallel and waits for them to finish computation. When computation is finished, data is sent back to ARM CPU. Bcrypt implemen-

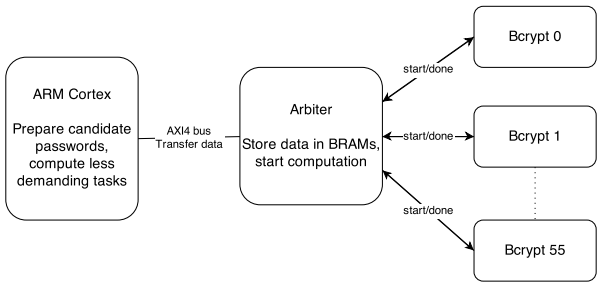

Figure 1: bcrypt implementation on ZedBoard

tation in reconfigurable logic is fast and uses small portion of available resources, which allows for high number of bcrypt instances running in parallel. For example, in a Zynq 7020 device, if four BRAMs are used to store S-boxes for four bcrypt instances and one BRAM is used to store other data (P-box, expanded key, salt and cost) for four bcrypt instances this equals to a maximum of 112 bcrypt instances running in parallel. This memory layout fully utilizes the available BRAM resources (140 BRAMs) because all available ports of true dual-port BRAMs are used on every clock cycle of Blowfish encryption round: eight lookups from BRAMs holding S-boxes and two lookups from BRAM holding P-boxes and other data for both instances.

However, this design was initially unreliable because of a combination of Zynq PS core voltage drop and insufficient decoupling from PL main voltage supply. On ZedBoard and on Parallella board, both of these are provided by the same 1.0V voltage regulator output. Our ZedBoard was rebooting right away, and on Parallella (its revision with Zynq 7020) we measured (via Zynq's own ADC) a voltage drop from 960 mV (somewhat low) to 890 mV (unacceptable). To overcome this limitation, we modified ZedBoard adding a wire going from C357 on the back of the board (near the relevant voltage regulator) to C217 near Zynq (C217 is a capacitor among those decoupling VCCPINT, the PS core voltage), thereby reducing the resistance of this specific path. We also added a 10 nF capacitor (which might or might not have mattered) and a couple of 470 uF electrolytic capacitors (one wasn't quite enough per our testing, albeit possibly in terms of ESR rather than capacitance) in parallel with C217. With these changes, the 112 bcrypt instances design could finally work long enough for us to take voltage measurements, and the lowest we could capture with a multimeter was over 970 mV on C217 (of course, this would have been more appropriately measured with a high frequency oscilloscope). The next hurdle was heat, which we solved by adding a 12V 0.08A 40x40mm cooling fan onto the Zynq heatsink (the fan looks huge compared to the heatsink!) and powering it from one of the pins of J21 "current sense" connector and a ground pin in one of the Pmod connectors. With these modifications in place, the 112 bcrypt instances design became stable and can be used reliably (on this specific board). Disconnecting the fan temporarily so that we could use J21 for its intended purpose, we measured (via J21) that ZedBoard's power consumption increases by around 1.5W when we load and start to use this bitstream on bcrypt cost 12 hashes (thus, deliberately achieving the maximum power consumption by keeping communication delays to a minimum). Another 1W is consumed by the fan.

The described 112 instances design artificially splits computation across two cycles because only two lookups from S-boxes can be done in a single clock cycle. But if S-boxes for a single bcrypt instance are stored in two BRAMs instead of in one, it is possible to fetch all four 32-bit values in a single clock cycle by performing eight S-box lookups from four BRAMs. This results in reduction of maximum number of instances running in parallel to 56, limited by available BRAM. However, these two designs have the same performance because halving the number of parallel instances is compensated by twice faster computation. Limitation of both designs is communication overhead, which impacts performance at lower cost settings, but becomes negligible at higher cost settings. Communication overhead comes in part from transferring 56 (or 112) 4KB large sets of S-boxes filled with initial values from ARM cores to reconfigurable logic and it can be avoided by storing those initial values in unused portions of available BRAM. Since each BRAM block on Zynq can hold 4 KB of data, the BRAM blocks used to hold other than S-box data (whose size is 164 B) are mostly empty and can be used to store initial values of S-boxes. However, this design is unstable at more than 28 bcrypt instances (in our testing) because of physical limitations of ZedBoard (presumably, insufficient decoupling between PS and PL power despite of our modifications so far).

To overcome these limitations, experiments were conducted on a bigger device from the Zynq family, ZC706 board with Zynq 7045 reconfigurable logic [15]. Unfortunately, this device was not much more reliable at high instance counts than ZedBoard without modification was. However, it was possible to port 56 instances design, which was not working on ZedBoard to ZC706. Maximum number of concurrent instances is 216 (limited by the available BRAM), but it is not reliable. The highest instance count working reliably is 196. Apart from this, we used Zynq 7045 device and tested our unstable ZedBoard design to obtain performance figures for

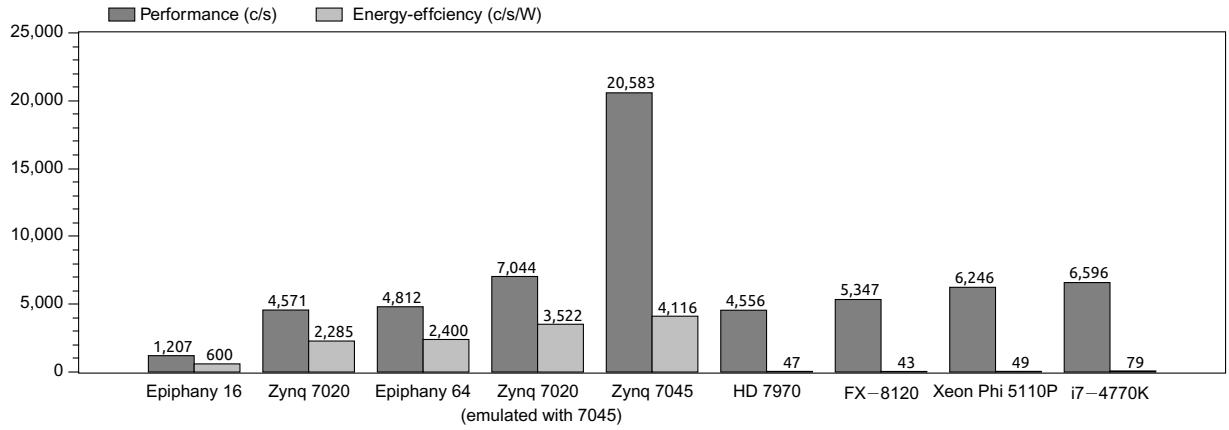

Figure 2: Performance and Energy-Efficiency of various platforms

design that should have worked on ZedBoard if it were more reliable. To overcome limitations of Zynq boards, work is underway on porting the design to ZTEX quad Spartan-6 LX150 boards [18].

4 Experimental Results

Figure 2 gives the comparison of performance and energy efficiency of bcrypt implementations for different platforms including our energy-efficient implementations: Epiphany 16, Epiphany 64 and ZedBoard. In addition we implemented and measured the performance and energy efficiency on commodity multicore CPUs represented by the four–core i7-4770K and eight–core FX-8120. Finally we also performed experiments with the GPU implementation on the HD 7970 graphic card and many integrated core architecture with Intel's Xeon Phi 5110P processor. Dark–gray bars represent mere performance in cracks per second, while light–gray bars give performance per energy consumption, i.e. efficiency in cracks per seconds per Watt. Our low–cost parallel platforms are comparable to or outperform multicores and GPUs in terms of performance in cracks per second and their energy efficiency is far better. ZedBoard outperforms all devices in terms of performance and energy efficiency. Figure 4 compares Epiphany manycore with x86 CPUs fabricated in same (or nearly the same) technology. The performance of Epiphany chips is comparable to x86 CPUs while energy-efficiency is tens of times higher due to its low power consumption of only 2 Watts. Results for Epiphany show linear scalability in performance with the increasing number of cores (from 16 to 64). Energy–efficiency scales linearly due to fabrication process upgrade from 65nm to 28nm (Epiphany 16 to Epiphany 64). Although most of the communication overhead on Zynq 7020 was eliminated by storing initial S-box values in reconfigurable logic, some data

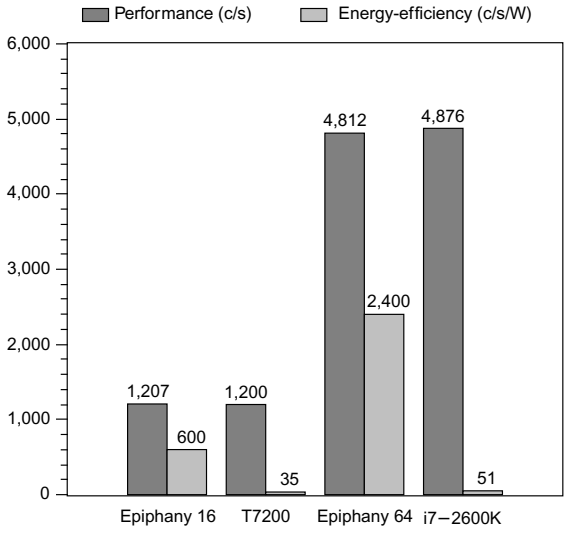

Figure 4: Epiphany vs x86

needs to be transferred. This has impact on Zynq 7020's performance at lower cost settings. To see how it compares to other platforms when communication overhead becomes negligible, benchmarks were run for higher cost settings (8, 10 and 12). Results are summarized in Table 1. At higher cost settings, Zynq 7020 performance is comparable to performance achieved on high end multicore CPUs.

Except from direct performance comparison at higher cost setting, it is possible to compare theoretical performance for cost 5 derived from measured performance for cost 12. Calculation is done based on difference in the number of Blowfish encryptions for different cost settings (Equation 1):

$$c/s = \frac{(2^{12} * 1024 + 585)}{(2^5 * 1024 + 585)} * performance_{12} \qquad (1)$$

Blowfish encryption is executed $9 + 512$ times before

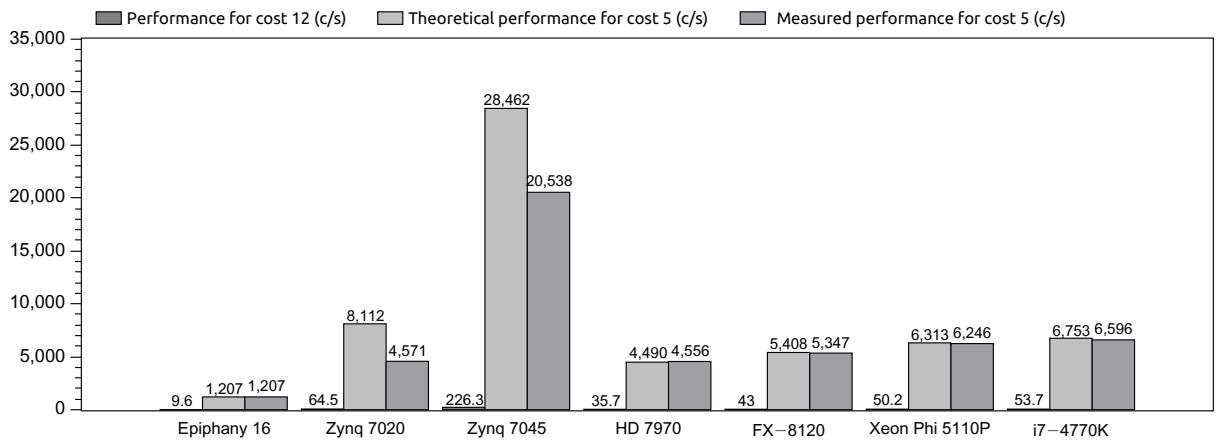

Figure 3: Theoretical performance for cost 5 derived from performance for cost 12

Cost / Device	12	10	8	5
Epiphany 16	9.64 c/s	38.7 c/s	151.3 c/s	1207 c/s
Zynq-7020	64.83 c/s	253.1 c/s	932.6 c/s	4571 c/s
Zynq-7045	226.3 c/s	888.6 c/s	3371 c/s	20538 c/s
HD 7970	35.76 c/s	142.9 c/s	569.2 c/s	4556 c/s
FX-8120	42.93 c/s	171.2 c/s	680.2 c/s	5275 c/s
Xeon Phi 5110P	50.18 c/s	200.7 c/s	800.8 c/s	6285 c/s
i7-4770K	53.67 c/s	214.2 c/s	852.8 c/s	6615 c/s

Table 1: Performance across different platforms for higher cost settings

most costly loop (Algorithm 1, line 2) and 64 or 192 times after it (Algorithm 2, lines 3, 4). We use number 64 because during attacks, only first 64 bits of hash are computed most of the time, which gives 585 Blowfish encryptions done outside most costly loop. In the single iteration of the most costly loop 1024 Blowfish encryptions are done. This ratio is multiplied with measured performance for cost 12 to derive theoretical performance for cost 5. Figure 3 shows results of this derivation. Zynq 7020 on ZedBoard is not only comparable to high end desktop CPUs and GPUs but it outperforms them in terms of performance.

Another important aspect of platform comparison is platform cost. Figure 5 shows how different platforms compare to each other in cracks per second per dollar metrics. We use the various platforms' prices at introduction for our comparison. System prices include board prices for Epiphany 16[1], Epiphany 64[2] and ZedBoard[3], estimated system price of $300 for system needed to run a CPU including motherboard, RAM and PSU. Chip and device prices are prices of devices themselves, not including any other components. As to PCIe cards, on one hand it is possible to install up to 8 per system, but on the other hand system prices are typically way higher than our estimate of $300 for a bare-bones CPU-only system. Due to variance in possible GPU and Xeon Phi whole system prices, we use device prices only. Our energy-efficient platforms are in the same price category as the CPUs we compare them against, and are a lot cheaper than Xeon Phi and some of the GPUs. When looking at c/s/$ performance considering system price, energy-efficient platforms outperform desktop CPUs. However, when considering chip and device prices, CPUs outperform Epiphany 16 while Zynq 7020 has the best performance. Even though CPUs perform comparable to Epiphany manycore, when attacking bcrypt hashes energy-efficiency plays a role. It is not just cost of the hardware that is important but also cost of the power consumption as well as cooling equipment because attacking thousands of bcrypt hashes can take days, months, or years even with focused wordlists.

[1]Current price is $119

[2]Intended price, it is not available on the market

[3]Academic price is either $299 or $319

5 Theoretical Peak Performance Analysis

Apart from measured performance in c/s and energy-efficiency in c/s/W it is possible to compare various hardware platforms using theoretical c/s figure derived from the platform characteristics. This figure can be calculated

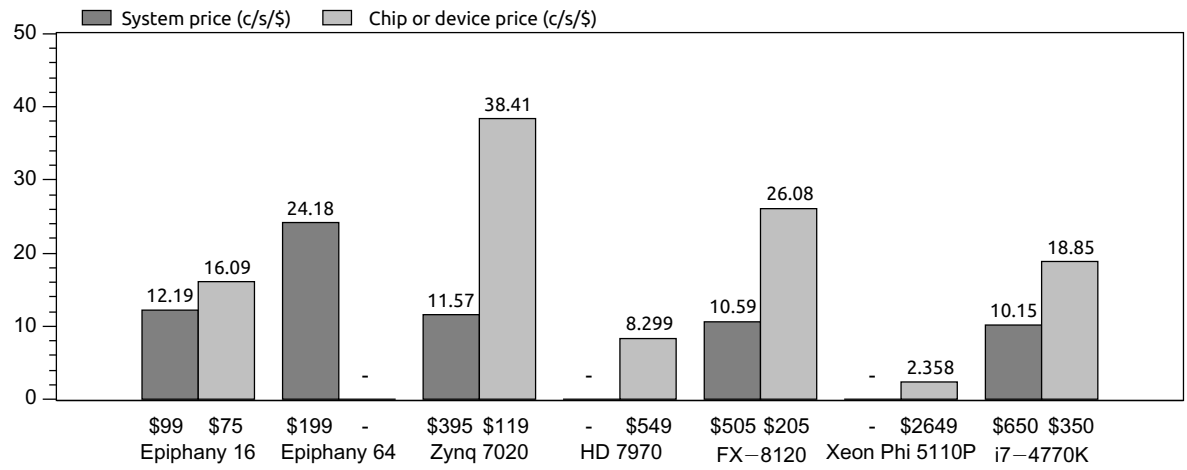

Figure 5: Performance in cracks per second per dollar of various platforms

using Equation 2:

$$c/s = \frac{N_{ports} * f}{(2^{cost} * 1024 + 585) * N_{reads} * 16} \quad (2)$$

where N_{ports} denotes number of available read ports to local memory or L1 cache (depending on the underlying hardware platform) and N_{reads} denotes the number of reads per Blowfish round (either 4 or 5 depending on whether reads from P-boxes go from one of those read ports we've counted or from separate storage such as registers). Expression $2^{cost} * 1024 + 585$ is the number of Blowfish block encryptions in bcrypt hash computation and 16 is the number of Blowfish rounds per encryption. Clock rate (f in Hz) is used instead of Blowfish block encryption rate because the encryptions can be interleaved, after which point the speed is limited by the local memory read rate. In some cases, the speed is limited by the local memory latency if local memory size is not sufficient for optimal interleaving to hide the latency. For example, modern x86 CPUs do up to two reads and one write to/from L1 cache per cycle per core. On Sandy Bridge and Ivy Bridge, these reads are up to 128-bit each. On Haswell, they're up to 256-bit each. However, in bcrypt, they are used as 32-bit either way. This is why Haswell's AVX2 gather loads do not improve bcrypt performance: they are limited to using these same two read ports.

6 Related Work

Related work in terms of optimizing hash algorithms is mostly focused on heavily optimized CPU implementations. This holds true for bcrypt algorithm which has been implemented on desktop CPUs [10]. Apart from CPU implementations, there are OpenCL and OpenMP implementations supporting GPUs and many-core devices [9]. As to hardware bcrypt implementations, another team has been working independently to improve our initial results for ZedBoard [8], and succeeded at that [5]. However, results presented in this work improve our initial results by a factor of 9 and results presented in [5] by a factor of 2.25. Work presented in [5] shows that there is much potential for improving the clock rate, which will likely result in speeds that are much higher yet.

Apart from bcrypt, there exist implementations of other hash algorithms in hardware such as UNIX Crypt hardware password cracker [12]. Encryption algorithms including Blowfish on which bcrypt is based and DES targeting different hardware platforms have been reported in [7, 14, 6, 11]. Rather than pure performance, our work focuses on energy efficiency and cost of underlying platforms as well.

7 Conclusion

In this paper, we have shown that there are low-power parallel platforms capable of exploiting bcrypt peculiarities to achieve decent performance and much better energy efficiency when compared to multicore desktop CPUs and GPUs. Higher energy efficiency also enables higher density. Higher density means more chips per board and more boards per system, which gives an attacker more cracking power for the same cost when using energy-efficient low cost hardware instead of more commonly used CPUs and GPUs.

Future work includes optimizations on Zynq 7020 and Zynq 7045, using Parallella board with multiple Epiphany 64 chips and porting implementation to ZTEX boards with four Spartan 6 FPGAs [18]. Future optimizations on Zynq 7020 and Zynq 7045 include reducing communication overhead, which limits performance at lower cost settings, and increasing clock rate. Once optimized, performance for bcrypt implementation on

Zynq 7020 should be comparable to high end CPUs for both low and high cost settings. Parallella board supports up to 64 Epiphany chips. If using 64 Epiphany 64 chips this sums up to 4096 cores. Based on scalability of implementation between E16 and E64, theoretical performance of Parallella boards with 64 E64 chips is $\sim$ 300000 c/s. Apart from this, future work on Parallella includes using both Epiphany and Zynq 7020 at once. With four Spartan 6 FPGAs, estimated performance for bcrypt implementation on ZTEX board is tens of thousands of c/s, which will outperform currently available CPUs and GPUs.

Existing energy-efficient bcrypt implementations and future work with very promising performance estimates have shown that it is possible to achieve decent performance in executing bcrypt on hardware. What is worrying is the fact it can be achieved with low cost hardware, which outperforms multicore CPUs and GPUs in terms of performance and energy efficiency. This shows that bcrypt will not remain secure forever and new, more advanced and attack resistant password hashing algorithms have to be devised.

References

[1] ADAPTEVA. Epiphany Architecture Reference. http://adapteva.com/docs/epiphany_arch_ref.pdf, 2013.

[2] ADAPTEVA. Parallella Computer Specifications. http://www.parallella.org/board/, 2013.

[3] ADAPTEVA. Parallella Reference Manual. http://www.parallella.org/docs/parallella_manual.pdf, 2013.

[4] DESIGNER, S., AND MARECHAL, S. Password security: past, present, future. http://www.openwall.com/presentations/Passwords12-The-Future-Of-Hashing/, 2012.

[5] F. WIEMER, R. Z. Speed and Area-Optimized Password Search of bcrypt on FPGAs.

[6] FOUNDATION, E. F. EFF DES cracker. http://en.wikipedia.org/wiki/EFF_DES_cracker, 1998.

[7] KIRAN, L. K., ABHILASH, J. E. N., AND KUMAR, P. S. FPGA Implementation of Blowfish Cryptosystem Using VHDL.

[8] MALVONI, K., AND DESIGNER, S. Energy-efficient bcrypt cracking. http://www.openwall.com/presentations/Passwords13-Energy-Efficient-Cracking/, 2013.

[9] OPENWALL. John the Ripper password cracker. http://www.openwall.com/john/.

[10] OPENWALL. Modern password hashing for your software and your servers. http://www.openwall.com/crypt/.

[11] PATEL, M. C. R., GOHIL, P. N. B., AND SHAH, P. V. FPGA - hardware based DES and Blowfish symmetric cipher algorithms for encryption and decryption of secured wireless data communication.

[12] POPPITZ, M. FPGA Based UNIX Crypt Hardware Password Cracker. http://www.sump.org/projects/password/, 2006.

[13] PROVOS, N., AND MAZIÈRES, D. A Future-Adaptable Password Scheme. *Proceedings of the FREENIX Track:1999 USENIX Annual Technical Conference* (1999).

[14] SIMMLER, H., KUGEL, A., MANNER, R., VIEIRA, A., GALVEZ-DURAND, DE ALCANTARA, F., J.M.S., AND ALVES, V. Implementation of cryptographic applications on the reconfigurable FPGA coprocessor microEnable.

[15] XILINX. Xilinx Zynq-7000 All Programmable SoC ZC706 Evaluation Kit. http://www.xilinx.com/products/boards-and-kits/EK-Z7-ZC706-G.htm.

[16] XILINX. XUP ZedBoard. http://www.xilinx.com/support/university/boards-portfolio/xup-boards/XUPZedBoard.html.

[17] XILINX. Zynq–7000 All Programmable SoC Family of Reconfigurable Devices.

[18] ZTEX. USB-FPGA Module 1.15. http://www.ztex.de/usb-fpga-1/usb-fpga-1.15.e.html.