

## Energy-efficient bcrypt cracking

Katja Malvoni

(kmalvoni at openwall.com)

Solar Designer

(solar at openwall.com)

Openwall

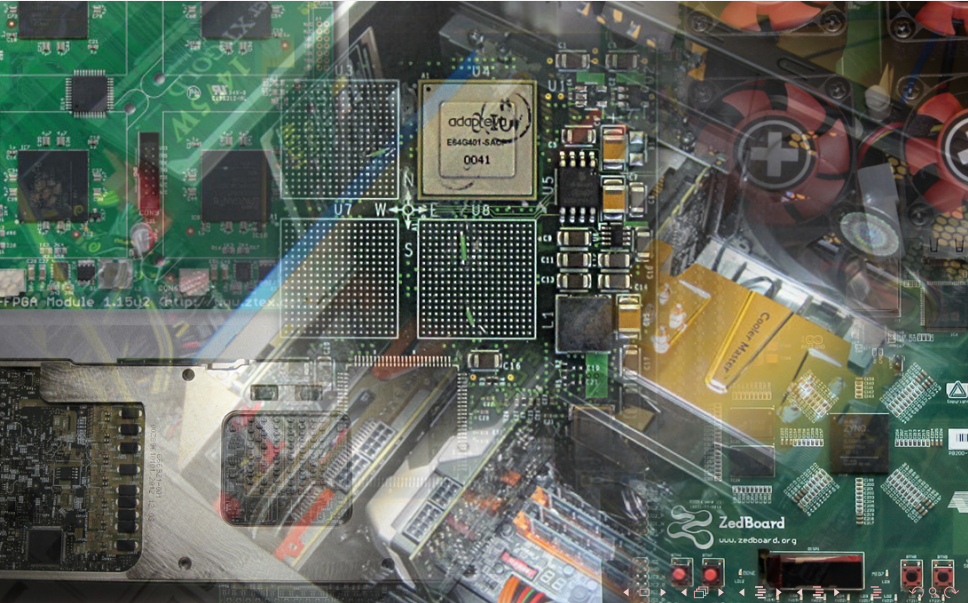
<http://www.openwall.com>

December 2, 2013

# Motivation

- Bcrypt is:
  - ▶ Slow
  - ▶ Sequential
  - ▶ Designed to be resistant to brute force attacks and to remain secure despite hardware improvements
- You could almost think why even bother optimizing

But



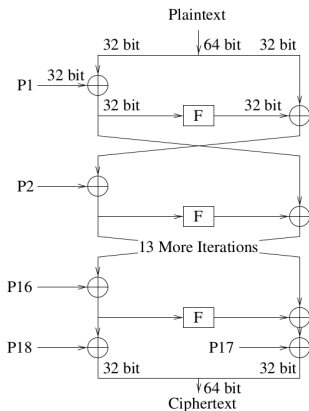
# Outline

- 1 Bcrypt
- 2 Implementation on different hardware
  - Parallella/Epiphany
  - ZedBoard
  - Xeon Phi
  - Haswell
- 3 Power consumption
- 4 Demo
- 5 Future work
- 6 Takeaways

# Bcrypt

- Based on Blowfish block cipher
- Expensive key setup
- User defined cost setting
  - ▶ Cost setting between 4 and 31 inclusive is supported
  - ▶ Cost 5 is traditionally used for benchmarks for historical reasons
  - ▶ All given performance figures are for bcrypt at cost 5
  - ▶ Current systems should use higher cost setting
- Pseudorandom memory accesses
- Memory usage
  - ▶ 4 KB for four S-boxes
  - ▶ 72 B for P-box

# Blowfish encryption



- 64-bit input block
- Feistel network
- Pseudorandom memory accesses
  - ▶ 32-bit loads from four 1 KB S-boxes initialized with digits of number  $\pi$

$$R_i = L_{i-1} \oplus P_i \quad (1)$$

$$L_i = R_{i-1} \oplus F(R_i) \quad (2)$$

$$F(a, b, c, d) = ((S_1[a] + S_2[b]) \oplus S_3[c]) + S_4[d] \quad (3)$$

Niels Provos and David Mazieres, "A Future-Adaptable Password Scheme", The OpenBSD Project, 1999

# EksBlowfish

## Ekspensive key schedule Blowfish

---

**Algorithm 1** EksBlowfishSetup(cost, salt, key)

---

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

---

- Order of lines 4 and 5 is swapped in implementation

# bcrypt

---

**Algorithm 2**  $\text{bcrypt}(\text{cost}, \text{salt}, \text{pwd})$ 

---

- 1:  $\text{state} \leftarrow \text{EksBlowfishSetup}(\text{cost}, \text{salt}, \text{key})$
  - 2:  $\text{ciphertext} \leftarrow \text{"OrpheanBeholderScryDoubt"}$
  - 3:  $\text{repeat}(64)$
  - 4:      $\text{ciphertext} \leftarrow \text{EncryptECB}(\text{state}, \text{ciphertext})$
  - 5:  $\text{return Concatenate}(\text{cost}, \text{salt}, \text{ciphertext})$
-



# Outline

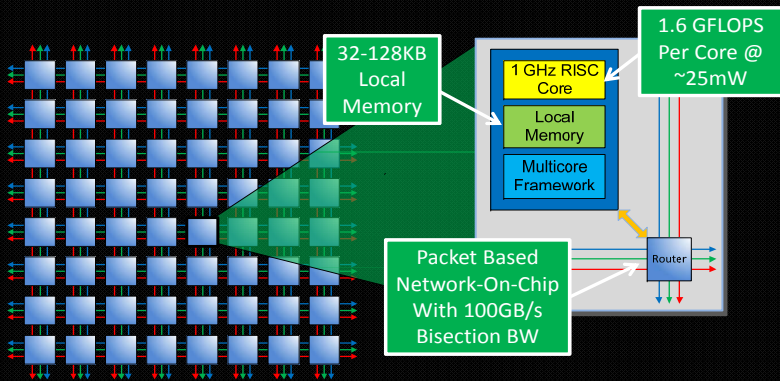
- 1 Bcrypt
- 2 Implementation on different hardware
  - Parallella/Epiphany
    - ZedBoard
    - Xeon Phi
    - Haswell
- 3 Power consumption
- 4 Demo
- 5 Future work
- 6 Takeaways

# Architecture

## Epiphany

- 16/64 32-bit RISC cores operating at up to 1 GHz/800 MHz
  - ▶ Chips used in our testing operate at 600 MHz
- Pros
  - ▶ **Energy-efficient** - 2 W maximum chip power consumption
  - ▶ 32 KB of local memory per core
  - ▶ 64 registers
  - ▶ FPU can be switched to integer mode
  - ▶ Dual-register (64-bit) load/store instructions
  - ▶ Ability for cores and host to directly address other cores' local memory
- Cons
  - ▶ FPU in integer mode can issue only add and mul instructions
  - ▶ Only simple addressing modes
    - Index scaling would be helpful for S-box lookups

# The Epiphany Coprocessor



Coprocessor for  
ARM/x86 Host

<20pJ / FLOP !

MIMD/Task-Parallel  
Accelerator

# Implementation

## One bcrypt instance per core

- Bcrypt algorithm in C
  - ▶ First working multi-core code: 822 c/s
  - ▶ All code moved to core local memory: 932 c/s
- C + variable-cost portion of eksBlowfish in Epiphany asm
  - ▶ Goal is to make use of dual-issue
  - ▶ First attempt slower than compiler generated code
  - ▶ Reschedule instructions to make use of dual-issue
  - ▶ And again
  - ▶ ...
  - ▶ Finally: 976 c/s

# Implementation

## Two bcrypt instances per core

- Two instances because:
  - ▶ Instructions executed on FPU have 4 cycles latency
  - ▶ Single bcrypt instance doesn't have enough parallelism to hide this
  - ▶ Adding second instance brings sufficient amount of parallelism down to instruction level to hide those latencies
- Bcrypt algorithm in C
  - ▶ 947 c/s
  - ▶ Preload P-boxes: 996 c/s
- C + variable-cost portion of eksBlowfish in Epiphany asm
  - ▶ 1194 c/s
  - ▶ Transfer keys only when changed: 1207 c/s
    - When cracking multiple hashes, with different salts
- And 227 e-mails on the john-dev mailing list

# Implementation

## Epiphany asm

```

.macro BF2_2ROUND_B P1, P2, P3, P4
    and tmpa1, L0, c1
    lsr tmpa3, L0, 0xe
    and tmpa3, tmpa3, c2
    lsr tmpa4, L0, 0x16
    and tmpa4, tmpa4, c2
    imul tmpa1, tmpa1, c3
    ldr tmpa3, [S01, +tmpa3]
    ldr tmpa4, [S00, +tmpa4]
    lsr tmpa2, L0, 6
    and tmpa2, tmpa2, c2
    iadd tmpa3, tmpa4, tmpa3
    ldr tmpa2, [S02, +tmpa2]
    ldr tmpa1, [S03, +tmpa1]
    lsr tmpb4, L1, 0x18
    eor R0, R0, \P1
    eor tmpa3, tmpa2, tmpa3
    imul tmpb4, tmpb4, c3
    and tmpb1, L1, c1
    lsr tmpb3, L1, 0xe
    and tmpb3, tmpb3, c2
    iadd tmpa3, tmpa3, tmpa1
    imul tmpb1, tmpb1, c3
    ldr tmpb3, [S11, +tmpb3]
    ldr tmpb4, [S10, +tmpb4]
    lsr tmpa1, L1, 6
    and tmpa1, tmpa1, c2
    eor R0, R0, tmpa3
    iadd tmpb3, tmpb4, tmpb3
    ldr tmpa1, [S12, +tmpa1]
    ldr tmpb1, [S13, +tmpb1]
    lsr tmpa4, R0, 0x18
    eor R1, R1, \P3
    eor tmpb3, tmpa1, tmpb3
    imul tmpa4, tmpa4, c3

    and tmpa1, R0, c1
    lsr tmpa3, R0, 0xe
    and tmpa3, tmpa3, c2
    iadd tmpb3, tmpb3, tmpb1
    imul tmpa1, tmpa1, c3
    ldr tmpa3, [S01, +tmpa3]
    ldr tmpa4, [S00, +tmpa4]
    lsr tmpa2, R0, 6
    and tmpa2, tmpa2, c2
    eor R1, R1, tmpb3
    iadd tmpa3, tmpa4, tmpa3
    ldr tmpa2, [S02, +tmpa2]
    ldr tmpa1, [S03, +tmpa1]
    lsr tmpb4, R1, 0x18
    eor L0, L0, \P2
    eor tmpa3, tmpa2, tmpa3
    imul tmpb4, tmpb4, c3
    and tmpb1, R1, c1
    lsr tmpb3, R1, 0xe
    and tmpb3, tmpb3, c2
    iadd tmpa3, tmpa3, tmpa1
    ldr tmpb3, [S11, +tmpb3]
    ldr tmpb4, [S10, +tmpb4]
    imul tmpb1, tmpb1, c3
    lsr tmpa1, R1, 6
    and tmpa1, tmpa1, c2
    iadd tmpb3, tmpb4, tmpb3
    eor L0, L0, tmpa3
    ldr tmpa1, [S12, +tmpa1]
    eor L1, L1, \P4
    ldr tmpb1, [S13, +tmpb1]
    eor tmpb3, tmpa1, tmpb3
    add tmpb3, tmpb3, tmpb1
    eor L1, L1, tmpb3
.endm

```

# Implementation

## Epiphany asm

```
ldrd P00, [ctx0]
ldrd P02, [ctx0, +0x1]
ldrd P04, [ctx0, +0x2]
ldrd P06, [ctx0, +0x3]
ldrd P08, [ctx0, +0x4]
ldrd P010, [ctx0, +0x5]
ldrd P012, [ctx0, +0x6]
ldrd P014, [ctx0, +0x7]
ldrd P016, [ctx0, +0x8]
ldrd P10, [ctx1]
ldrd P12, [ctx1, +0x1]
ldrd P14, [ctx1, +0x2]
ldrd P16, [ctx1, +0x3]
ldrd P18, [ctx1, +0x4]
ldrd P110, [ctx1, +0x5]
ldrd P112, [ctx1, +0x6]
ldrd P114, [ctx1, +0x7]
ldrd P116, [ctx1, +0x8]
```

loop2:

```
eor L0, P00, L0
eor L1, P10, L1
BF2_2ROUND_B P01, P02, P11, P12
BF2_2ROUND_B P03, P04, P13, P14
BF2_2ROUND_B P05, P06, P15, P16
BF2_2ROUND_B P07, P08, P17, P18
BF2_2ROUND_B P09, P010, P19, P110
BF2_2ROUND_B P011, P012, P111, P112
BF2_2ROUND_B P013, P014, P113, P114
BF2_2ROUND_B P015, P016, P115, P116
eor tmpa2, R0, P017
strd tmpa2, [ptr0], +0x1
eor tmpa3, R1, P117
strd tmpa3, [ptr1], +0x1
mov R0, L0
mov R1, L1
mov L0, tmpa2
mov L1, tmpa3
sub tmpa4, end, ptr0
bgtu loop2
```

# Implementation

## Summary

- Two bcrypt instances per core
- Optimized in assembly
- Dual-issue
  - ▶ Integer ALU
  - ▶ FPU in integer mode
- Integrated into John the Ripper

```
kmalvoni@linaro-ubuntu-desktop:~/JohnTheRipper/run$ sudo -E LD_LIBRARY_PATH=$LD_LIBRARY_PATH ./john -test -format=bcrypt-parallella  
Benchmarking: bcrypt-parallella, OpenBSD Blowfish ("2a$05", 32 iterations) [Parallella]... DONE  
Raw: 1207 c/s real, 1207 c/s virtual
```

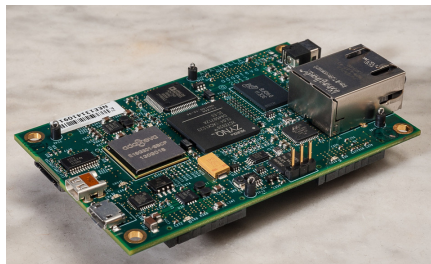
```
linaro@linaro-ubuntu-desktop:~/JohnTheRipper/run$ sudo -E LD_LIBRARY_PATH=$LD_LIBRARY_PATH ./john -test -format=bcrypt-parallella  
Benchmarking: bcrypt-parallella, OpenBSD Blowfish ("2a$05", 32 iterations) [Parallella]... DONE  
Raw: 4812 c/s real, 4812 c/s virtual
```



# Performance

## Epiphany 16

- 1207 c/s
- $\sim 600$  c/s per Watt
- We achieved 3/4th of the per-MHz per-core speed of a full integer dual-issue architecture



Parallella Board. Photo (c) Adapteva, reproduced under the fair use doctrine

# Performance

## Epiphany 64

- 4812 c/s
- $\sim 2400$  c/s per Watt for E64 chip
- $\sim 1000$  c/s per Watt for Parallella board with E64
  - ▶ When not yet using the ARM cores and FPGA PL for computation
- Scalability
  - ▶  $4812/1207 = 3.987\times$  faster
  - ▶ 99.7 % efficiency
    - $4812/1207/4 = 0.9967$

Epiphany 16

```
#define EPIPHANY_CORES 16
```

Epiphany 64

```
#define EPIPHANY_CORES 64
```

# ZedBoard + FMC with E16 or E64 - prototyping

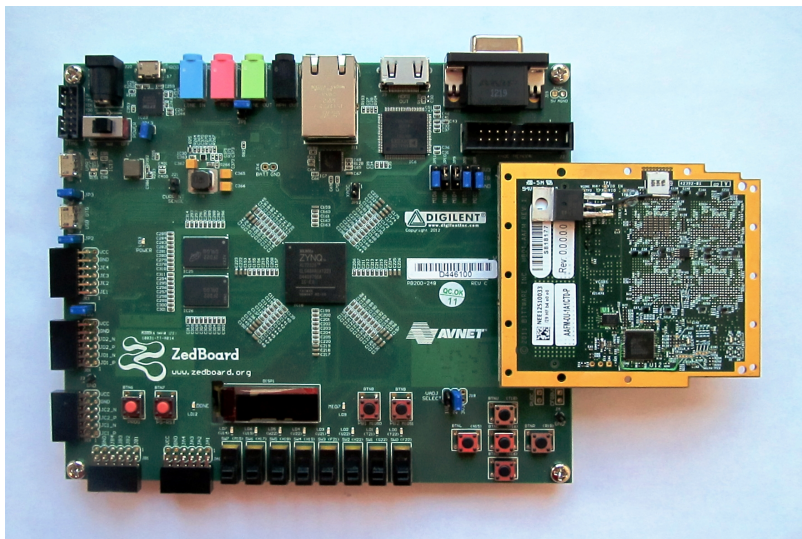


Photo (c) Adapteva, used with permission

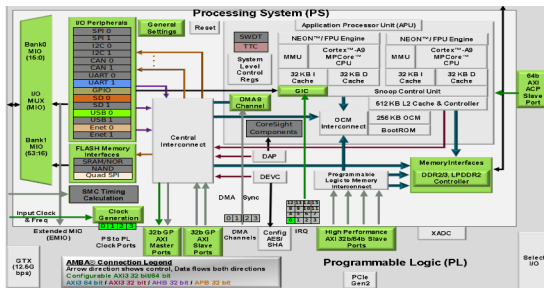
# Outline

- 1 Bcrypt
- 2 Implementation on different hardware
  - Parallella/Epiphany
  - ZedBoard
  - Xeon Phi
  - Haswell
- 3 Power consumption
- 4 Demo
- 5 Future work
- 6 Takeaways

# Architecture

## Zynq 7020

- Dual ARM Cortex-A9 MPCore
  - ▶ 667 MHz
  - ▶ 256 KB on-chip memory
- Advanced low power 28nm programmable logic
  - ▶ 85 K logic cells
  - ▶ 560 KB of block RAM

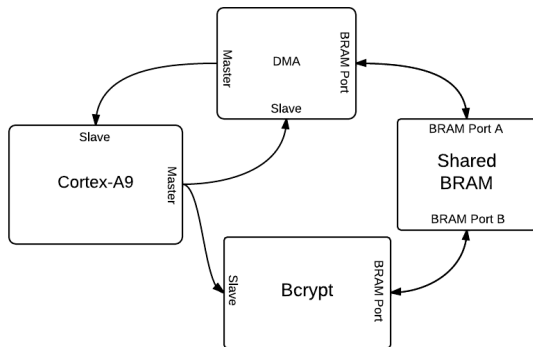


Zynq diagram. Screenshot from Xilinx Platform Studio, reproduced under the fair use doctrine

# Implementation

## CPU/FPGA communication

- General Purpose Master AXI interface
- Accelerator Coherency Port Slave AXI interface
- DMA from on-chip Memory to BRAM



# Implementation

## 5 cycles per round

- Non-optimal implementation: 13.9 c/s
  - ▶ S-boxes are stored in shared BRAM so only one port is used for load
- Copy S-boxes to another dual port BRAM and use both ports: 23.5 c/s
- Per-cycle summary
  - ▶ Cycle 0: initiate 2 S-box lookups
  - ▶ Cycle 1: wait
  - ▶ Cycle 2: initiate other 2 S-box lookups, compute tmp
  - ▶ Cycle 3: wait
  - ▶ Cycle 4: compute new L, swap L and R
- 14 cores fit: 311 c/s

# Implementation

## 2 cycles per round

- Use two dual port BRAMs
  - ▶ Two S-boxes in one BRAM, two in the other
  - ▶ Load all 4 needed values in one cycle
  - ▶ Single core speed: 79 c/s
  - ▶ 14 cores still fit: 780 c/s
    - With no overhead, theoretical performance would be  $79 \cdot 14 = 1106$  c/s
    - With bcrypt cost setting above 5, efficiency is higher
- Per-cycle summary
  - ▶ Cycle 0: compute new R; swap L and R; initiate 4 S-box lookups
  - ▶ Cycle 1: wait



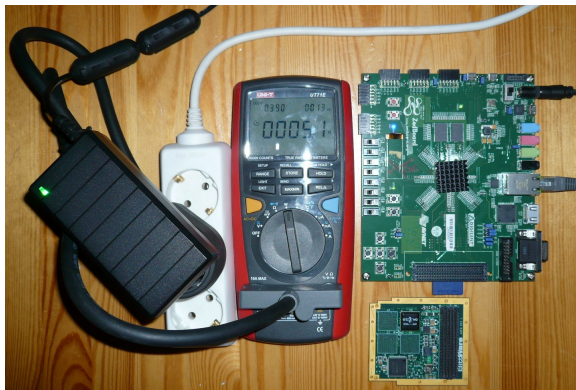
# Implementation

## Summary

- Only most costly loop (algorithm 1, lines 3, 4 and 5) implemented in FPGA
- 2 cycles per one Feistel network round
- 14 bcrypt cores at 100 MHz
- Utilization
  - ▶ Register: 19%
  - ▶ LUT: 90%
  - ▶ Slice: 98%
  - ▶ RAMB36E1: 11%
  - ▶ RAMB18E1: 5%
  - ▶ DSP48E1: 6%
  - ▶ BUFG: 3%

# Performance

- 780 c/s
- $\sim 400$  c/s per Watt for FPGA
- 153 c/s per Watt for ZedBoard



# Implementation

## Room for improvement

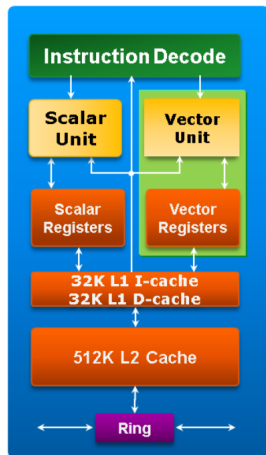
- Sanity-check and reduce LUT and Slice utilization
  - ▶ Current utilization feels insane
- Reduce support logic
  - ▶ Currently wastes 1/3 of FPGA resources
  - ▶ Longest path limits clock rate
- Consider redesigns
  - ▶ Multiple bcrypt instances per state machine
  - ▶ One S-box per BRAM, leaving a port for initialization
    - May reduce LUT utilization by not needing a mux
- Improve CPU/FPGA communication
  - ▶ Can be made asynchronous

# Outline

- 1 Bcrypt
- 2 Implementation on different hardware**
  - Parallella/Epiphany
  - ZedBoard
  - Xeon Phi**
  - Haswell
- 3 Power consumption
- 4 Demo
- 5 Future work
- 6 Takeaways

# Architecture

## 5110P



- 60 Cores
- 1.053 GHz
- Max TDP 225 W
  - ▶ In special cases can go up to 245 W
- SIMD vector instructions
- Each core has 512 bit wide vector processor unit (VPU)
  - ▶ 16 32-bit integer operations per clock cycle
  - ▶ Vector mask registers
- Each core supports 4 hardware threads

# Implementation and Performance

- Using scalar units for computation
  - ▶ John the Ripper with OpenMP build: 6246 c/s
    - Using native OpenMP programming model (not offload)
  - ▶ John the Ripper with OpenCL build: 6017 c/s
- Using VPU for computation
  - ▶ C with MIC intrinsics, masked gather loads
  - ▶ Two bcrypt instances per thread, 240 threads: 4147 c/s
  - ▶ Other combinations of number of instances per thread and number of threads per core result in even lower performance



# Outline

- 1 Bcrypt
- 2 Implementation on different hardware**
  - Parallella/Epiphany
  - ZedBoard
  - Xeon Phi
  - Haswell**
- 3 Power consumption
- 4 Demo
- 5 Future work
- 6 Takeaways

# Haswell

## i7-4770K

- AVX2 supports 256-bit (8 x 32-bit) integer gather loads
- Bcrypt code written by Steve Thomas
  - ▶ 8 bcrypt instances per thread
  - ▶ 8 threads on 4 cores: 4186 c/s
  - ▶ Over 64 KB per core, but we only have 32 KB L1 data cache
- Can we improve performance by staying in cache?
  - ▶ Running 4 threads with original code which only slightly exceeds L1 data cache size: 3888 c/s
  - ▶ Different memory layout and using 7 instead of 8 instances to stay below 32 KB: 3519 c/s
- Existing non-AVX2 code in John the Ripper: 6595 c/s
  - ▶ 2 bcrypt instances per thread, 8 threads
- Apparently, Haswell's gather loads are just slow; maybe a future CPU will do better



# Outline

- 1 Bcrypt
- 2 Implementation on different hardware
  - Parallella/Epiphany
  - ZedBoard
  - Xeon Phi
  - Haswell
- 3 Power consumption
- 4 Demo
- 5 Future work
- 6 Takeaways

# CPU performance

CPU	Cores/threads	Lithography	Performance	Efficiency	Chip power estimate	TDP	Idle to load delta	Full system
T7200	2c/2t 2.0 GHz	65 nm	1200 c/s <sup>1,2</sup>	35 c/s/W	34 W	34 W	36 W	44 W
Q8400	4c/4t 2.66 GHz	45 nm	3484 c/s <sup>2</sup>	40 c/s/W	88 W	95 W	54 W	120 W
i7-2600K	4c/8t 3.4+ GHz	32 nm	4876 c/s <sup>2</sup>	51 c/s/W	95 W	95 W	72 W	139 W
FX-8120	4m/8t 3.1+ GHz	32 nm	5347 c/s <sup>2</sup>	43 c/s/W	124 W	125 W	140 W <sup>3</sup>	250 W <sup>4</sup>
i7-4770K	4c/8t 3.5+ GHz	22 nm	6595 c/s	79 c/s/W	84 W	84 W		
2x E5-2670	16c/32t 2.6+ GHz	32 nm	16900 c/s <sup>2</sup>	73 c/s/W	230 W	2x 115 W	217 W	606 W <sup>4</sup>

System power consumption includes PSU overhead (typically 5% to 20%), hence deltas may exceed CPUs' TDP

<sup>1</sup>1216 c/s for short runs, ~1180 c/s after CPU heats up

<sup>2</sup>Modified John the Ripper 1.8.0 code to introduce 3x interleaving (3 bcrypt instances per thread), instead of 1.8.0's default of 2x interleaving

<sup>3</sup>After CPU fan fully spins up, consuming extra 12 W

<sup>4</sup>Includes other devices (idle)

# GPU and MIC performance

Device	Core/memory <sup>1</sup>	Lithography	Performance	Efficiency	Device power estimate	TDP	Idle to load delta	Full system <sup>2</sup>
GTX TITAN	902+/1652 MHz <sup>3</sup>	28 nm	813 c/s	6 c/s/W	135 W	250 W	120 W	510 W
GTX 570	1600/1000 MHz <sup>4</sup>	40 nm	1224 c/s	9 c/s/W	131 W	219 W	137 W	247 W
HD 7970	925/1375 MHz	28 nm	4556 c/s	47 c/s/W	96 W	250 W	95 W	205 W
HD 7970	1225/1075 MHz <sup>5</sup>	28 nm	6008 c/s	53 c/s/W	113 W	N/A	115 W	225 W
Xeon Phi 5110P	1053/1250 MHz	22 nm	6246 c/s	49 c/s/W	128 W	225/245 W	38 W	428 W
HD 7990	2x 1000/1500 MHz	28 nm	2x 4269 c/s <sup>6</sup>	46 c/s/W	185 W	375+ W	176 W	566 W

bcrypt is an extremely poor fit for current GPUs, and vice versa

<sup>1</sup> GDDR5 memory, so effective memory speed is 4x higher than shown

<sup>2</sup> Includes other devices (idle)

<sup>3</sup> Zotac GeForce GTX TITAN AMP! Edition, vendor's overclocking

<sup>4</sup> Palit GTX 570 Sonic Platinum, vendor's overclocking


<sup>5</sup> Extreme overclocking, only possible due to bcrypt heavily under-utilizing the GPU (as it has to because of limited local memory)

<sup>6</sup> The per-chip c/s rate regression from HD 7970 is because of the newer Catalyst version as required to support the HD 7990

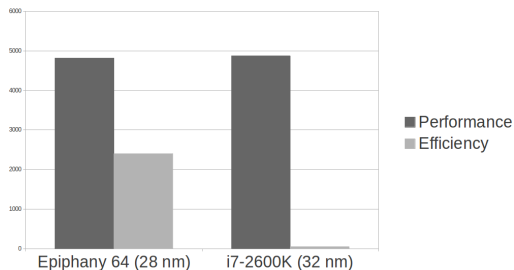
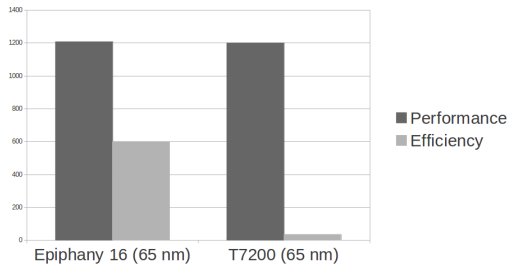
# Energy-efficient platforms

Platform	Cores	Lithography	Performance	Efficiency	Chip power estimate	TDP	Idle to load delta	Full system
Epiphany 16	16c 600 MHz	65 nm	1207 c/s	600 c/s/W	2 W	2 W	1.3 W	9.1 W <sup>1</sup>
Epiphany 64	64c 600 MHz	28 nm	4812 c/s	2400 c/s/W	2 W	2 W		
Zynq-7020	14c 100 MHz	28 nm	780 c/s	400 c/s/W	2 W		1.0 W	5.1 W <sup>2</sup>

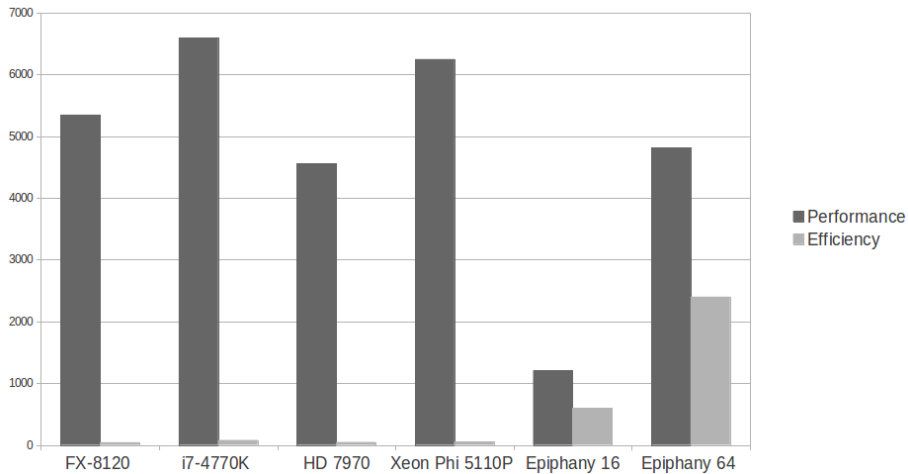
<sup>1</sup>ZedBoard + FMC with E16 chip and glue logic, together simulating a Parallella board. The actual Parallella board should consume less power

<sup>2</sup>Same ZedBoard, but with the FMC disconnected and FPGA bitstream replaced 

# Epiphany vs x86



# Performance and efficiency comparison



# Outline

- 1 Bcrypt
- 2 Implementation on different hardware
  - Parallella/Epiphany
  - ZedBoard
  - Xeon Phi
  - Haswell
- 3 Power consumption
- 4 Demo
- 5 Future work
- 6 Takeaways

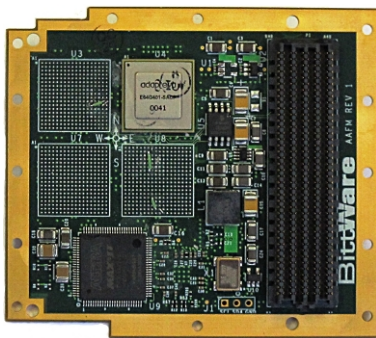
# Outline

- 1 Bcrypt
- 2 Implementation on different hardware
  - Parallella/Epiphany
  - ZedBoard
  - Xeon Phi
  - Haswell
- 3 Power consumption
- 4 Demo
- 5 Future work**
- 6 Takeaways



# Parallella/Epiphany

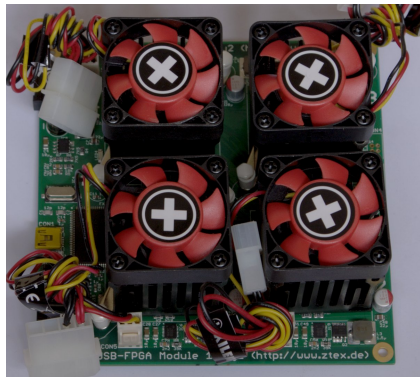
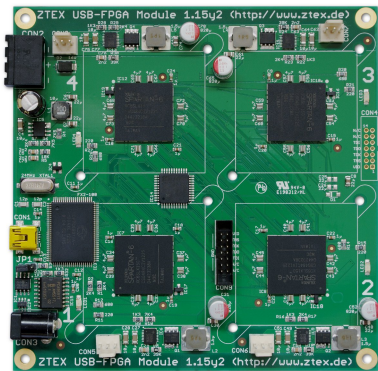
- Chip to chip links for integrating up to 64 chips on a single board
- Scalability of current implementation is promising
- $64 * 64 = 4096$  cores with theoretical performance of 300000 c/s



FMC with E64 chip and pads for 3 more such chips. Photo (c) Adapteva, used with permission

# FPGA

- ZedBoard optimizations
- Targetting bigger FPGAs
- Targetting multi-FPGA boards



ZTEX Board. Photo (c) ZTEX, reproduced under the fair use doctrine

# Outline

- 1 Bcrypt
- 2 Implementation on different hardware
  - Parallella/Epiphany
  - ZedBoard
  - Xeon Phi
  - Haswell
- 3 Power consumption
- 4 Demo
- 5 Future work
- 6 Takeaways**

# Takeaways

- Many-core low power RISC platforms and FPGAs are capable of exploiting bcrypt peculiarities to achieve higher energy-efficiency
- Higher energy-efficiency enables higher density
  - ▶ More chips per board, more boards per system
- It doesn't take ASICs to improve bcrypt cracking energy-efficiency by a factor of 10+
  - ▶ Although ASICs would do better yet

# Thanks

- Sayantan Datta
- Steve Thomas
- Parallella project
- Google Summer of Code

# Questions



kmalvoni at openwall.com