ProgPOW Hardware Audit Report

Bob Rao

Final Report to

Ethereum Cat Herders, Ethereum Foundation and Bitfly

September 6 2019

Outline

- Executive summary
- Audit goals and expectations
- Executive summary details
- Key ASIC strategies
- Breakdown of ProgPOW algorithm, DRAM latencies, memory hardness
- ASIC approaches 1, 2 - integrating DAG memory and logic on chip
- Some preliminary system cost estimates for proposed ASICs
- Compare GPUs cards and current Ethash ASICs on the market
- ASIC approach 3 - light evaluation method (calculate DAG entries on the fly)
- Conclusions
- Background material

Executive summary

- The ProgPOW algorithm works well to mitigate conventional ASIC strategies:
  - These focus only on implementing the compute engines in silicon (memory still in off chip DRAM), on the premise that domain specific circuits are always more energy efficient than general purpose circuits.
  - Random memory access latencies and energy expended to move data from DRAM to the compute engines are the same for GPU or conventional ASIC.
  - Memory hardness which is shared by both ProgPOW and Ethash (with ProgPOW being 2X that of Ethash) dominates over compute in determining Energy/Hash (E/H)
  - ProgPOW vastly increases the complexity of the inner loop compute and optimizes it to the GPU datapath, thereby further mitigating E/H improvements possible with conventional ASICs.
- However, there is a looming threat to memory hard algorithms in general as massive amounts of memory can be brought very close to compute logic due to advancements in Moore's law and 3D/2.5D packaging:
  - 10/7nm processes provide up to 25Mbits/mm^2 of SRAM and 100MTransistors/mm^2 enabling vast amounts of fast SRAM and compute logic to be integrated on chip at an economically feasible die size and cost.
  - The SRAM/Transistor densities keep increasing exponentially over time due to Moore's law
  - 3D/2.5D packaging brings memory and compute very close together to the point of being almost on-chip:
    - A combination of monolithic and stacked approaches can optimize massive lateral and vertical connectivity
  - The energy expended to move data from DRAM to the compute logic (> 3pJ/bit) dominates and this is reduced by >>10X if the memory is integrated with the compute logic.
  - With sufficient on-chip memory available, ProgPOW ASICs with << 0.1X E/H over GPUs can be built

Audit goals and expectations

1. The expected effects of ProgPOW on the security of Ethereum vis-à-vis: Security of the algorithm, attack surface, cost of 51% attack, and other security risks that may result from a change from Ethash to ProgPOW:
   - Two notable observations include a) The ProgPOW memory hardness is 2X more difficult than Ethash 2) The DAG and the Cache from which it is generated are identical for ProgPOW and Ethash – so it not expected that any additional security risks are introduced here
   - The Keccak algorithm is different from Ethash "The implementation is a variant of SHAKE with width=800, bitrate=576, capacity=224, output=256, and no padding" and should be checked. The hardware audit did not look at this
2. ***ProgPOW meeting the goal of ASIC resistance: Known methods to speed up the calculation of the hash function, length of time it would take to create a ProgPOW ASIC (if R&D begins immediately), and expected efficiency gains from the first generation of said ASICs (Focus of the Hardware Audit)***
   - **Current Ethash Asics on the market are demonstrating ~1.6X Hashrate/Watt over GPUs. ProgPOW ASICs will be more complex to design but are still fundamentally limited by memory hardness and the improvements over GPUs are expected to be less than for Ethash**
   - **The development/tapeout/validation and packaging of advanced ASICs on 10nm+ processes will cost $20M+ and take 1+ year to develop → can be done if there is a 150 day ROI to miners**
   - **The bulk of this hardware audit focuses on new architectures, enabled by Moore's law and advanced packaging, that will mitigate memory hardness**
3. Identify any potential advantages and disadvantages that ProgPOW would present in comparison to Ethash in terms of changes to the network, "fair mining" and evaluate any potential uneven distribution:
   - If ProgPOW is deployed, the network will lose the contribution to hashrate from dedicated ASIC hardware
   - While GPUs can be reprogrammed from Ethash to ProgPOW, current Ethash ASIC hardware cannot be reused for ProgPOW and will need to be scrapped

Executive summary - details

- The ProgPOW algorithm extends the current Ethash algorithm for Ethereum mining in two ways:
  - Vastly increase the computational parts and match them to the GPU datapath, to attempt to saturate GPU processors to preclude any advantages for ASICs
  - Increase memory hardness further by doubling the data fetched from the DAG to saturate the memory bandwidth utilization between the GPU and the DRAM*****
- When comparing GPUs and ASIC performance, the only meaningful metric is Energy (J)/Hash [E/H]or Watts/Hashrate [W/HR] (lower the better)
- By offloading only the computation part of ProgPOW to an ASIC (Keccak, inner loop, random math, cache, memory controller) a modest reduction in E/H is feasible (1.6X demonstrated by currently available hardware on Ethash) as, by far, memory accesses dominate
  - Since every memory access in the inner loop is random, every read to the DAG incurs the random-access latency (~40ns)** which is completely independent of the memory Bandwidth or the computation engine (GPU or an ASIC)
  - The GPU attempts to issue thousands of parallel threads corresponding to different hashes to hide this latency but due to conflicts the memory bandwidth cannot be fully saturated***
  - In this sense ProgPOW works well to mitigate conventional ASIC strategies which focus only on implementing the compute engines in silicon on the premise that domain specific circuits are always more energy efficient than general purpose circuits****. The random memory access latencies and the energy expended to move data from DRAM to compute are the same for GPU or ASIC.
- However, there is a looming threat to memory hard algorithms in general as massive amounts of memory can be brought very close to compute logic due to advancements in Moore's law and 3D/2.5D packaging:
  - 10/7nm processes provide up to 25Mbits/mm^2 of SRAM and 100MTransistors/mm^2 enabling vast amounts of fast SRAM and compute logic to be integrated on chip at an economically feasible die size and cost.
  - The SRAM/Transistor densities keep increasing exponentially over time due to Moore's law
  - 3D/2.5D packaging brings memory and compute very close together to the point of being almost on-chip
  - The energy expended to move data from DRAM to the compute logic (> 3pJ/bit) dominates and this is reduced by >>10X if the memory is integrated with the compute logic.
  - With sufficient on-chip memory available, ProgPOW ASICs with << 0.1X E/H over GPUs can be built

* DRAM in this report means any kind of modern DRAM including DDR4, GDDR5, GDDR5X, HB1, HB2 ..

** "A Performance & Power Comparison of Modern High-Speed DRAM Architectures", Li et al, MEMSYS, October 1–4, 2018, Old Town Alexandria, VA, USA © 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-6475-1/18/10. https://user.eng.umd.edu/~blj/papers/memsys2018-dramsim.pdf

*** https://medium.com/@ifdefelse/understanding-progpow-performance-and-tuning-d72713898db3

**** See for example "a New Golden Age for Computer Architecture", J.L. Hennessy & D.A. Patterson, Comm of the ACM, Feb 2019, Vol 62, No2, https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

ASIC strategy for Ethash and ProgPOW

- ASIC resistance primarily relies on memory hardness
- Memory hardness has always been the holy grail as it is extremely difficult to overcome with conventions DRAMs and random addressing:
  - The only way around this to integrate the RAM with the computational logic (SRAM, eDRAM, Stacked DRAM, RRAM, MRAM…)
  - In addition the DAG keeps increasing in size over time obsoleting memory cards when their capacity is exceeded (see backup)
  - ProgPOW (256B reads from DAG) has 2x memory hardness over Ethash (128B reads from DAG)
- Ethash focused on the memory hardness aspect:
  - Computations were quite simple (2 calls to Keccak hashes and 64 mixes using fnv1)
  - The GPU was underutilized and it was possible to build custom ASICs to offload the computation at lower power
- ProgPOW on the other hand has added a significant computation element to fully utilize and saturate the GPU logic with highly randomized loops
  - Computation can still be accelerated with an ASIC at lower power like with Ethash
  - But memory access strategy (same for GPU and ASIC) will set the limit

ASIC strategy

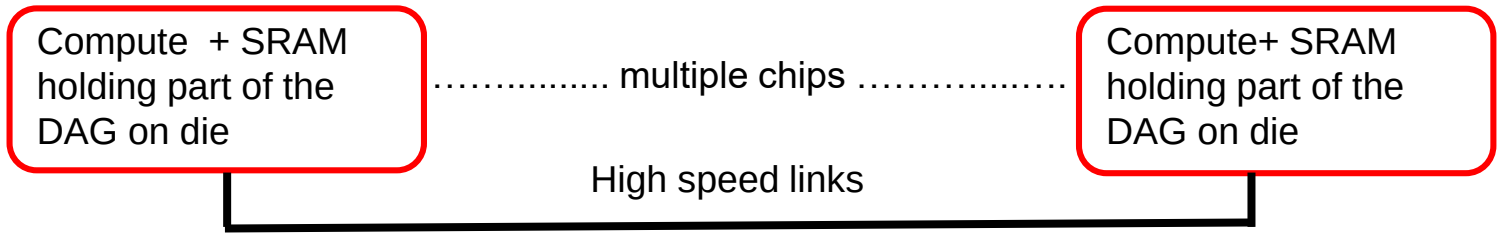
ASIC Accelerates Computation only
Keccak engine, ProgPOW inner loop, cache, memory controller

DRAM stores DAG
DAG refreshed every epoque

- Current state of the art: One ASIC for computation + Multiple DRAM chips side by side or stacked.
- Energy/Hash ~ .5X* of GPU/DRAM

ASIC accelerates computation and memory hardness (with 512MB SRAM):
Hardware to calculate DAG entry on the fly, Keccak engine, ProgPOW inner loop, hardware to refresh cache every epoque. Massively parallelize hashes

- Exploit 25Mbits/mm^2 SRAM and 100M transistors/mm^2 (~25MGate/mm^2) in 10nm process to integrate cache + logic on chip. Calculate DAG entry on the fly (light evaluation method)
- Energy/Hash < 0.1* X of GPU/DRAM



- Exploit 25Mbits/mm^2 SRAM and 100M transistors/mm^2 (~25MGate/mm^2) in 10nm process to store the entire DAG and compute on multiple chips
- Energy/Hash reductions over GPU/DRAM dependent on energy of high-speed links

ASIC accelerates computation and memory hardness (with > 3GB SRAM):
Complete DAG stored on die in SRAM, Keccak engine, ProgPOW inner loop. Massively parallelize hashes

- Exploit 100Mbits/mm^2 SRAM and 500M transistors/mm^2 (~125MGate/mm^2) in 3nm process around 2025 to store the entire DAG and compute on chip
- Energy/Hash << 0.1* X of GPU/DRAM

* Rough estimates only and need full simulations to validate

ASIC strategies to mitigate memory hardness

- Key strategy is to bring Memory (which holds the DAG or the Cache) much closer to the compute engine
  - Integration of SRAM main memory and computational logic on 10nm+ process technologies
  - 3-D stacking of SRAM memory and Logic
- Three possible approaches are proposed where SRAM holds either the entire DAG or the cache from which the DAG is created
  1. The entire DAG (3GB as of 7/1/2019) is loaded into the on-chip SRAM from where the ProgPOW inner loop compute hardware accesses it with very low energy (possible in 2025+)
  2. DAG + Compute are distributed over multiple chip which are interconnected with high speed links
  3. The on-chip SRAM only holds the Cache (~51MB as of 8/30/2019), while the compute hardware calculates every random DAG entry on the fly and feeds these entries in to the ProgPOW inner loop hardware. This is also referred to as “light evaluation method”
- ProgPOW inner loop, Keccak (see backup), FNV… can all be accelerated in custom silicon:
  - ProgPOW inner loop hardware implementation requires ~ 1 Million Gates* (~0.025mm^2 in 10nm process) → integrate 25 ProgPOW inner loop hardware units in 1mm^2 or 2500 in 100mm^2

* https://medium.com/@profheisenberg/everybody-knows-alus-are-relatively-tiny-circuits-a589d2de4cce

The ProgPOW algorithm

- The ProgPOW algorithm* was analyzed in detail to understand implications to memory hard ASICs.
- GPU implementation of the ProgPOW algorithm** was assessed to understand implications to memory hard ASICs
- A Keccak hash of the header + nonce to create a seed
  - The Keccak algorithm and random number generator (here used kiss99) can be offloaded to custom hardware
- Use the seed to generate initial mix data
  - Can be done in custom hardware
- Loop multiple times, each time hashing random loads and random math into the mix data
  - The inner loop is evaluated and the random math order is precomputed and compiled every 50 (ver0.9.2) blocks/10 minutes or 10(ver0.9.3) blocks/2 minutes on the host CPU. This code is then loaded into the GPU. Since this randomness is fixed for this period, it should be possible to hardware accelerate this part.
  - The most difficult part is the random DAG (Directed Acyclic Graph) access, which is hard to overcome with conventional DRAM
- Hash all the mix data into a single 256-bit value
  - Can be done in custom hardware
- A final Keccak hash is computed
  - Can be done in custom hardware
- When mining, this final value is compared against a hash_32 target

* https://github.com/ifdefelse/ProgPOW and

** https://medium.com/@ifdefelse/understanding-progpow-performance-and-tuning-d72713898db3

Typical latencies and bandwidths of modern DRAMs

- For random access reads like in Ethash and ProgPOW – the most important parameter is the ***latency*** because there is very little locality and every new address is completely random.
- Bandwidth is important for:
  - Sequential reads after the first random access
  - Pipelining many memory requests for parallel hashing
- The typical latency for back to back reads is tRP + tRCD +CL ~ 42nS or tRAS + tRP ~ 41nS

"A Performance & Power Comparison of Modern High-Speed DRAM Architectures", Li et al, MEMSYS, October 1–4, 2018, Old Town Alexandria, VA, USA © 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-6475-1/18/10.
https://doi.org/10.1145/3240302.3240315
https://user.eng.umd.edu/~blj/papers/memsys2018-dramsim.pdf

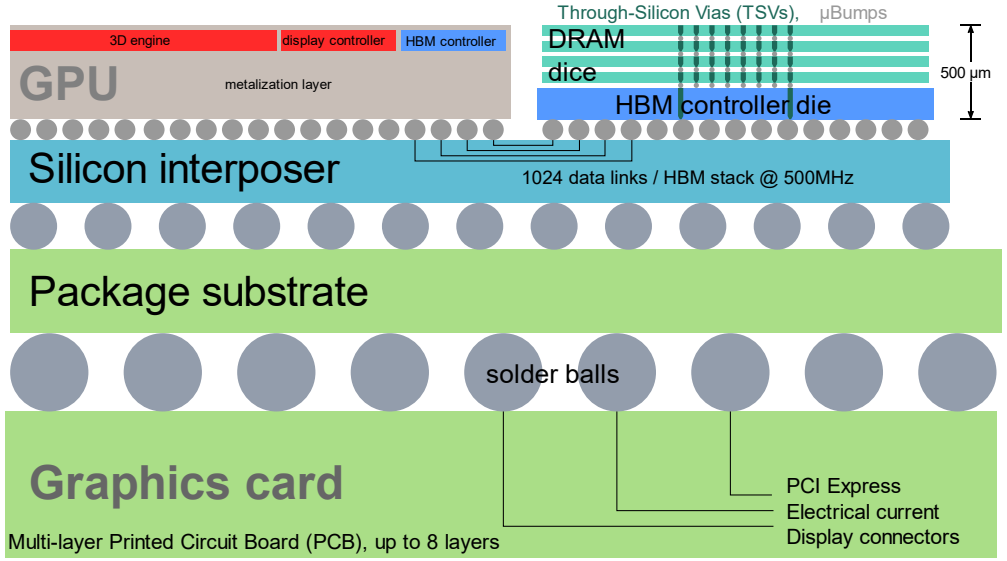
Memory hardness

- Memory hardness is provided by the limited random-access speeds of DRAM
- A significant amount of the power dissipation occurs due to movement of the data from within the DRAM, over the shared memory bus to the GPU/ASIC*
  - Typical DRAM energy dissipation is 3 – 4 pj/bit with much of it due to on die data movement
  - For example for HBM2 **1)** Activating a DRAM row incurs ~ 1.2 pj/bit **2)** Getting the data from the DRAM core to the IO pins takes a further ~2.6pj/bit **3)** Finally a further ~0.2 pj/bit is expended from getting the data from the DRAM IO pins to the GPU
  - An on-die memory will reduce this energy dissipation to << 0.1X (<<0.3pj/bit)

* O'Connor et al MICRO-50, October 14–18, 2017, Cambridge, MA, USA

Approach 1: Custom HBM stack

- An HBM stacks 4 – 8 DRAMs die on top of a controller logic die.
- This stack is then connected with a silicon interposer to the GPU
- The silicon interposer is then connected with TSVs to the package substrate
- It should be quite feasible to put the ProgPOW random computation/Cache engines in the HBM controller die:
  - A leading CMOS process (e.g. Intel 10nm or TSMC 7nm) attains ~100 Mtransistors/mm^2 or ~24M 2 input NAND Gates/mm^2*
  - A 14nm CMOS process attains 37.5 Mtransistors/mm^2 or ~10M Gates/mm^2**
  - A 22nm process attains ~ 15.3 Mtransistors/mm^2 or ~4M Gates/mm^2**



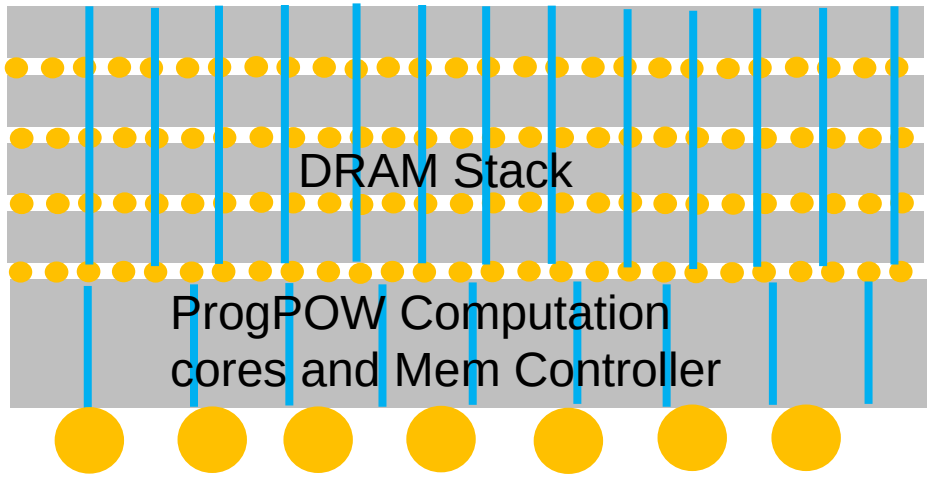
https://en.wikipedia.org/wiki/High_Bandwidth_Memory

* https://semiwiki.com/semiconductor-manufacturers/intel/7544-7nm-5nm-and-3nm-logic-current-and-projected-processes/

** https://spectrum.ieee.org/nanoclast/semiconductors/processors/intel-now-packs-100-million-transistors-in-each-square-millimeter

Approach 1: Custom HBM stack

- Assuming ~ 1 Million gates* for a ProgPOW custom ASIC computation core (500K gates for the SRAM, 400K gates for the register files and 100K gates for the Keccak, Mix (FNV1a), RNG (KISS99) and random math:
  - a huge number of such parallel cores can be placed on the logic controller die even with a 22nm process – although 10nm or 7nm is preferred due to lower power
  - See backup for custom implementation of Keccak
- Memory is now much closer to the logic hence energy can be up to 2X lower by reducing some on-DRAM data movement and IO power.
- Computation speedup depends on how many nonce calculations are done in parallel with the computation cores:
  - This will also depend on the power dissipation of the logic die and the thermal budgets for the DRAMs.
- Since HBM roadmap goes all the way to 64 GB/stack – this should be able to accommodate the continuing increase in DAG size
  - References ** and *** illustrate commercial announcements for 3D stacking



A possible implementation of a ProgPOW ASIC with a custom HBM stack

* https://medium.com/@profheisenberg/everybody-knows-alus-are-relatively-tiny-circuits-a589d2de4cce

** https://www.anandtech.com/show/14211/intels-interconnected-future-chipslets-emib-foveros

*** https://www.extremetech.com/computing/282137-intel-uses-new-foveros-3d-chip-stacking-technology-to-build-core-atom-on-the-same-silicon

Approach 2: Integrated logic and memory

- We can overcome memory hardness if we can store the entire SRAM DAG on chip without DRAM
- This is only possible if the process node provides adequate SRAM and transistor densities
- We can estimate if and when this will be possible, looking at Moore's law progression and SRAM density
  - True Moore's law scaling approximately doubles the transistor density on a cadence
  - The cadence used to be approximately two years – but is now longer – we will assume doubling every three years
- Currently process node naming is a bit murky (for example Intel's 10nm is equivalent to foundry 7nm in density) *. Based on this, we estimate approximate densities the industry is likely to deliver as a function of time in the following table.

Node	Year	SRAM Density (Mbits/mm^2)*	Transistor Density (MTr/mm^2)
Intel 10nm/Foundry 7nm*	2018/2019*	25***	100
Intel 7nm/Foundry 5/3nm*	2021/2022*	50***	250
NG Node 1 (2X projected)**	2024/2025**	100***	500**
NG Node 2 (2X projected)**	2027/2028**	200***	1000**

* Source: https://semiwiki.com/semiconductor-manufacturers/intel/7544-7nm-5nm-and-3nm-logic-current-and-projected-processes/

** Assume a Moore's law cadence (density doubling) every 3 years

*** SRAM density is estimated as ~ (1/SRAM cell size)*0.7 (array efficiency) and averaged for different suppliers

Approach 2: Integrated logic and memory

- The DAG size increases 8MB every epoch of 30,000 blocks – See backup foil DAG size
- The tables in the next foil illustrate two scenarios
  1. The entire SRAM DAG and logic are fabricated on a single large die or stacked
  2. The DAG and logic are split into 16 smaller die and interconnected with high speed links
  3. Moore's law ensures smaller die size, lower cost/GB over time
- To remain agnostic to the silicon foundry (and the confusing array of node names), we characterize Moore's law by the year (with 3-year cadence) and the expected SRAM and Transistor densities
- We matchup the SRAM densities in a given year with the DAG size for that year to estimate the die size needed and die cost for single and 16 die cases
- The logic transistor densities available today are ~ 100MTransistors/mm^2 (25 Mgates/mm^2 assuming 2 input NAND gate)
  - Assuming the computational logic for ProgPOW needs approximately 1M Gates*, the silicon real estate for the logic portion of ProgPOW will be negligible compared to the memory and will for now be ignored for now.

* https://medium.com/@profheisenberg/everybody-knows-alus-are-relatively-tiny-circuits-a589d2de4cce

Die costs for integrated logic and memory

WHOLE DAG ON ONE LARGE MONOLITHIC DIE						
Year	DAG Size (GB)	Wafer Cost* (assume 1.33x increase/node)	Die Size for Full DAG (mm^2)	No of Good die/wafer**	Die Cost $	Cost/GB
2018/2019*	2.78	$ 7,000.00	911	20	$ 351.58	$ 126.47
2021/2022*	4.64	$ 9,310.00	760	29	$ 321.47	$ 69.28
2024/2025**	6.49	$ 12,382.30	532	56	$ 221.29	$ 34.10
2027/2028**	8.34	$ 16,468.46	342	113	$ 145.69	$ 17.47

DAG SPLIT INTO 16 SMALL INDIVIDUAL DIE						
Year	DAG Size (GB)	Wafer Cost* (assume 1.33x increase/node)	Die Size for 1/16 Full Dag (mm^2)	No of Good Die/Wafer**	Die Cost $	Cost/GB
2018/2019*	2.78	$ 7,000.00	57	1058	$ 6.62	$ 38.09
2021/2022*	4.64	$ 9,310.00	48	1291	$ 7.21	$ 24.86
2024/2025**	6.49	$ 12,382.30	33	1903	$ 6.51	$ 16.04
2027/2028**	8.34	$ 16,468.46	21	3044	$ 5.41	$ 10.38

*Initial wafer cost assumption of $7000 in 2018/2019 and the subsequent 33% (1.33x) increase in wafer cost/node is based on extrapolation from published works (see for example Lisa Su Keynote IEDM 2017)

**No of good die of area A in a 300mm wafer=No of die/wafer $\{\pi \frac{(150-\sqrt{A})^2)}{\sqrt{A}}\}$ X Yield $\{e^{-D_0 A}\}$
$where\ D_0\ is\ the\ process\ defect\ density, here\ assumed\ to\ be\ 0.1\ defects/Cm^2$ see backup foil Yield and die cost for more details

System costs for integrated logic and memory

- A die that can hold the logic and entire DAG at any point in time becomes cost effective at around 2025 and beyond when the die cost is ~ $222 for a die size of 532mm^2 and a DAG size of 6.49GB. This is because Moore's law scaling is faster (exponential) versus DAG size growth (linear). Note however that large die have poor yield and hence much higher cost. Summarizing:
  - Silicon die cost - $222 + Package Assembly & Test cost ~ $10 + PCB cost ~ $25 + Heatsink, Miscellaneous and interface chips cost ~ $25 gives a total of $282
- If the total DAG memory is partitioned into 16 smaller die which are then interconnected with high speed links, then it becomes cost effective today. Each die costs $6.62, holds 2.78/16 GB = 173.75MB and has a die size of 57mm^2. The much smaller die size gives a huge increase in yield and a massive reduction in die cost. Summarizing:
  - Silicon die cost for 16 die at 16x$6.62 = $106 + Package, assembly & Test cost of ~16x$1/die = $16 + PCB cost ~ $25 + Heatsink, Miscellaneous and interface chips cost ~ $25 = ~ $172 total.
- For a conventional GPU boards: Cost is ~ 240
  - 8GB GDDR6 ~ $150
  - GPU silicon ~ $50
  - PCB ~ $25
  - Heatsink and miscellaneous ~$15
- By eliminating DRAM accesses, a significant reduction in the memory access energy (up to 10X) becomes possible:
  - However the chip to chip high speed communication links add additional energy and precise gains need to be simulated
  - A product in development from XTEND online AMAP Hyperminer (https://www.xtend.online/docs/Xtend_WP_Final.pdf) appears to use an approach like this
- One could expect ASICs based on these strategies to be produced if there's a 150 day ROI to the miners

Compare GPU cards and current ASICs for Ethash

MODEL	ETHASH				PROGPOW			
	Hashrate (MH/S)	Power (W)	Hasrate/W	Hashrate/W Overclocked	Hashrate (MH/s)	Power (W)	Hashrate/W	Hashrate/w Overclocked
Gigabyte RX460 4GB GDDR5 *	11	48	0.15	0.15	4.5	48	0.06	0.06
MSI RX470 8GB GDDR5 Armor *	20.6	65	0.2	0.24	9.7	90	0.07	0.07
XFX RX480 8GB GDDR5 GTR *	21	89	0.14	0.24	11.4	115	0.06	0.08
Sapphire RX580 8GB GDDR5 Nitro+ *	21	89	0.15	0.25	12.7	180	0.06	0.09
Zotac GTX1060 6GB GDDR5 SFF *	20.6	100	0.21	0.35	10.2	120	0.09	0.13
EVGA GTX1070 8GB GDDR5 SC *	27.7	132	0.21	0.36	13.7	140	0.1	0.12
EVGA 1070ti 8GB GDDR5 FTW *	27.7	138	0.2	0.36	13.9	170	0.08	0.13
PNY GTX1080ti 11GB GDDR5X Blower *	26	250	0.14	0.31	35.2	120	0.1	0.11
EVGA GTX1660ti 6GB GDDR6 SSC *	26	105	0.25	0.4	14.8	120	0.12	0.17
MSI RTX2080ti 11GB GDDR6 Duke *	51	250	0.2	0.33	35.7	250	0.14	0.17
Bitmain Altminer E3 **	190	800	0.24		Unknown		NA	
Inosilicon E10: ETHMaster ***	485	850	0.57		Unknown		NA	

- No ASICs announced for ProgPOW as of Sept 2019
- On Ethash the best ASIC performance (Hashrate/W) is ~ 1.6X over an overclocked GPU
- A product in development from XTEND online AMAP Hyperminer**** projects Ethash/ProgPOW Hashrates of 1040/520 MH/S for 200 W giving HR/W of 5.2/2.6 which is 16X/8X over GPU. This product uses a decentralized RAM integrated with the compute logic

* https://medium.com/altcoin-magazine/comprehensive-progpow-benchmark-715126798476
** https://www.asicminervalue.com/miners/bitmain/antminer-e3-190mh
*** https://www.asicminervalue.com/miners/innosilicon/a10-ethmaster-485mh
**** https://www.xtend.online/docs/Xtend_WP_Final.pdf

Approach 3: Light evaluation method (calculate DAG entries on the fly)

- The DAG (Directed Acyclic Graph) reads, are what makes Ethash/ProgPOW memory hard
- The current DAG size is about 3GB and is increasing 8MB every epoque (30K Blocks) or ~ 0.6 GB/year*
- The DAG is derived from a much smaller database called the “cache” which is currently ~50MB and is increasing 125 KB every epoque*:
  - Every epoque, the cache is regenerated from a seed, SHA3-512 hash and several rounds of RandMemoHash. All of these can be highly hardware accelerated
- Calculating a 512 DAG entry from a cache entry* involves the following calculations all of which can be accelerated in custom silicon
  - Keccak hash of an initial mix
  - Combining this initial mix with 256 random cache reads with the fnv algorithm.
  - Keccak hash of the final mix
- So the concept is this: rather than storing a 3 GB dataset on chip (hard to do today but may be possible in 2025+), store the much smaller 50MB “cache” in on chip SRAM (very doable), and use very fast compute cores to calculate the DAG entries on the fly
- The rest of the ProgPOW inner loop hardware can be highly accelerated and parallelized/pipelined, just like in other compute only ASICs
- Even if the time to compute a DAG entry on the fly is comparable to accessing it from DRAM, the energy cost can be 100X lower as everything is on chip

* https://github.com/ethereum/wiki/wiki/Ethash

Silicon implementation – area and energy

- In a 10/7 nm process 128 MB takes up ~ 44.7mm^2. These areas are modest and economically realizable.
  - 128MB ensures chip is not obsolete for at least 5 years as cache continues to grow (see backup for cache growth)
- There are 4 DAG reads per ProgPOW inner loop so we need to replicate 128MB blocks 4X to calculate all DAG reads in parallel:
  - 128MBx4 = 512MB → 178.8mm^2 → this is still economically viable on 10/7nm
  - Each 128MB SRAM blocks will have their own DAG calculation hardware (Keccak, FNV and RandMemoHash) and gate count is negligible compared to SRAM (100Mtransistors/mm^2 available)
- The ProgPOW inner loop hardware is estimated to take about 1M gates → we can put 25 of these in 1mm^2 → massive inner loop parallelism for simultaneous hashing becomes possible with low silicon area overhead.

Silicon implementation – memory

- A 64 MB L3 cache in a CPU typically has a latency of 10 – 20ns* → the larger the cache the larger the latency.
- For on the fly DAG entry calculation, the SRAM is a “main memory” and not a CPU cache (no TAG RAM and hit logic)
- A highly banked SRAM main memory with small (< 1MB) banks will be very fast
- A 4GHz clock frequency with a 1024 bit memory datapath and a memory latency of 1 clock cycle can calculate a DAG entry in ~45nS which is comparable to a DRAM access latency but at ~100X less energy
- Parallel/pipelined hash computations are all happening on chip and hence all energy costs to go off chip are eliminated giving massive energy gains.

* “Computer Architecture a quantitative approach”, J.L. Hennessy & D.A. Patterson, 6th edition, Elsevier 2019, Chapter 2 Memory Hierarch Design Page 79

Conclusions

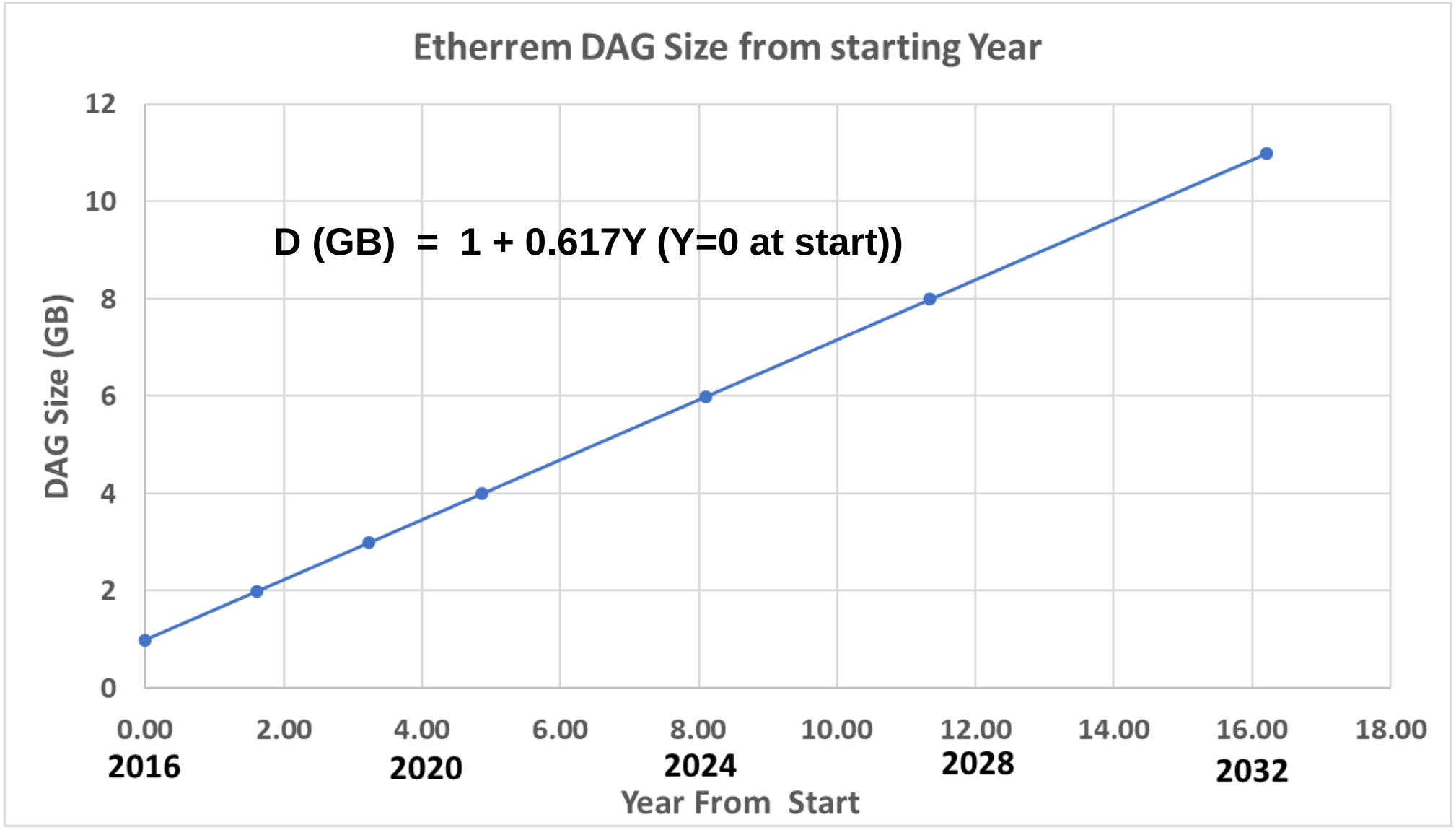
- The ProgPOW algorithm works well to mitigate conventional ASIC strategies which only address accelerating the computation side of this memory hard algorithm:
  - Current Ethash ASICs on the market are demonstrating ~1.6 X Hashrate/Watt over GPUs
- Memory hardness which is shared by both ProgPOW and Ethash dominates over compute in determining Energy/Hash (E/H)
- ProgPOW vastly increases the complexity of the inner loop compute and optimizes it to the GPU datapath, thereby further mitigating E/H improvements possible with conventional ASICs.
- However, there is a looming threat to memory hard algorithms in general as massive amounts of memory can be brought very close to compute logic due to advancements in Moore's law and 3D/2.5D packaging:
  - This bulk of this audit has explored possible approaches for these ASICs
- AI and graph workloads are facing similar memory issues and various companies are working on this:
  - It is entirely possible that new solutions will come from this AI segment

Acknowledgements

- I acknowledge the team of Least Authority for bringing up the light evaluation method to my attention

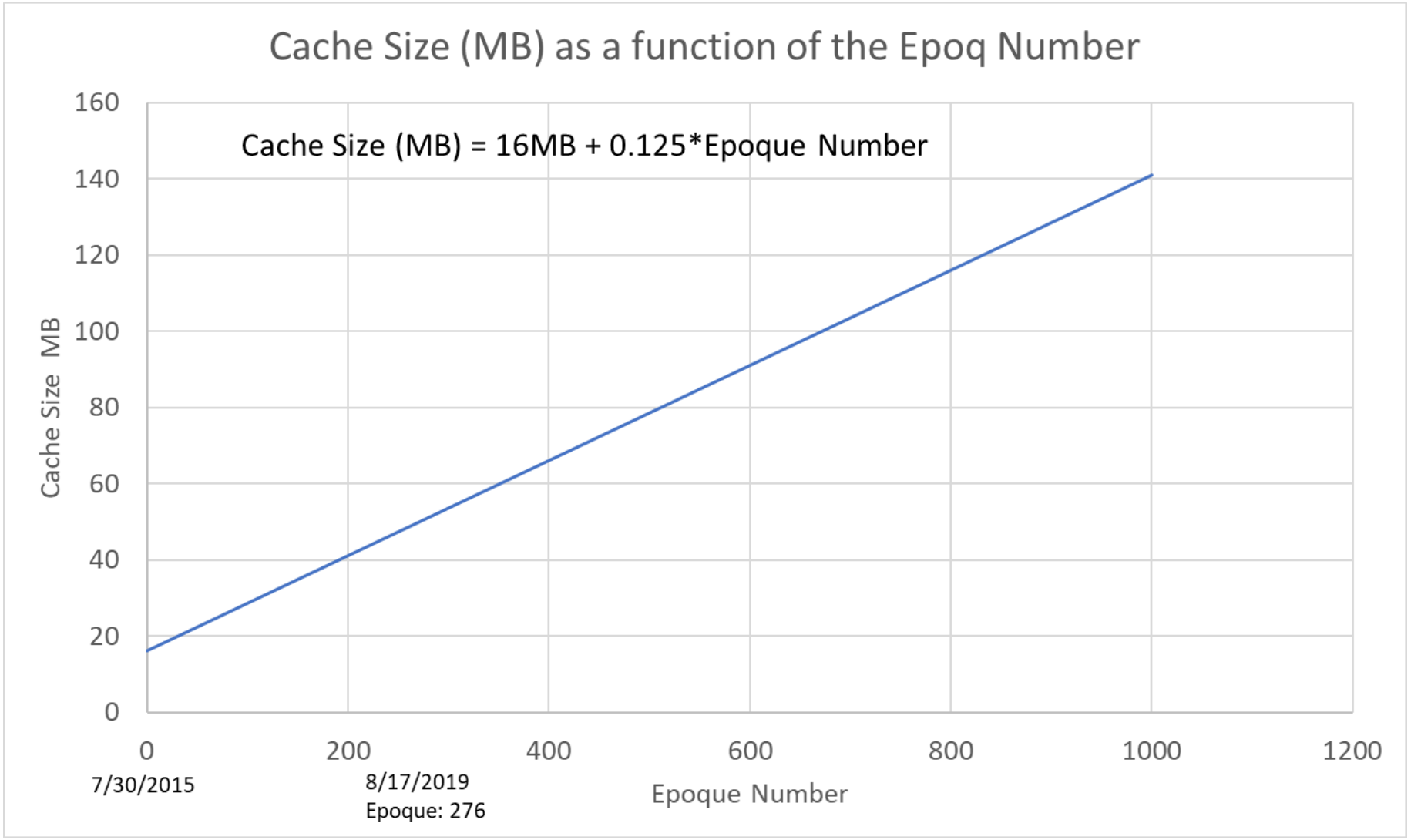
Background material

DAG size



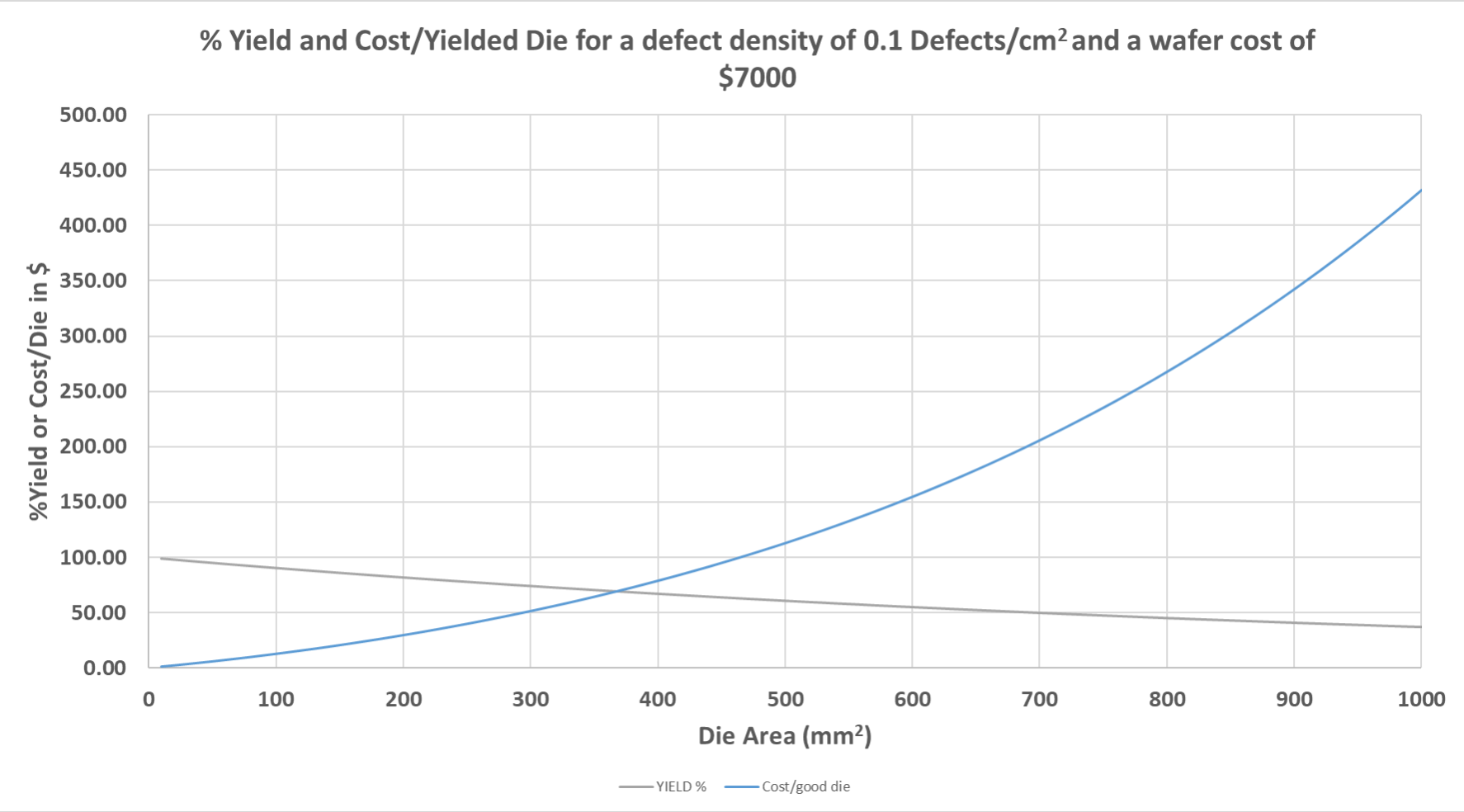
https://investoon.com/tools/dag_size

Cache size as a function of epoque number



- One epoque is 30,000 blocks
- On 7/17/2019 the block number was 8304504 in epoque 276
- Calculated from the definition list at https://github.com/ethereum/wiki/wiki/Ethash

Yield and die cost as a function of die area



- Assumptions
  - Wafer size = 300mm dia
  - Process defect density D_0= 0.1 defects/Cm^2
  - Poisson Yield Model
  - Wafer cost = $7000
- Then for a die area A, No of good die in a 300mm wafer is: No of die/wafer

$$\{\pi \frac{(150-\sqrt{A})^2)}{\sqrt{A}}\} \quad X \quad \text{Yield } \{e^{-D_0 A}\}$$

Published ASIC implementations of Keccak

Algorithm	HASH TYPE	ASIC/FPGA	F(MHz)	No of Rounds	Latency (Cycles)	Area K (Slices/GE)	Throughput (Gbits/sec)	Cycles Per Byte F/Throughput	Efficiency Mbps/GE
Keccak*	SMH	90nm ASIC	455	24	25	10.5	19.32	0.175	1.84
Keccak*	MMH	90nm ASIC	1695	24	121	23.2	74.41	0.17	3.21
Keccak**		65nm ASIC	1000	24			48	0.155	0.4571

- SMH is Single Message Hash and MMH is multi Message Hash
- It appears that Keccak implemented in an ASIC is about 100X faster than running on IA processors
- GPU based Keccak appears to be running at around in the range 1 – 3 (Cycles Per Byte) ***

* "Efficient Hardware Implementations of High Throughput SHA-3 Candidates Keccak, Luffa and Blue Midnight Wish for Single- and Multi-Message Hashing", Abdulkadir Akin et al Proceedings of the 3rd international conference on Security of information and networks Pages 168-177, September 07 - 11, 2010

** "A New High Throughput and Area Efficient SHA-3 Implementation", Ming Wong, Published in 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence Italy 27-30 May 2018

*** "GPU accelerated Keccak algorithm". Chanhui Wang et al, https://arxiv.org/abs/1902.05320:

**** "GPU implementation of Keccak hash function family", Pierre-Louis Cayrel et al, International Conference on Information Security and Assurance ISA 2011: pp 33-42

***** "Analysis of Keccak tree hashing on GPU architectures", J.M Lowden https://pdfs.semanticscholar.org/b1cd/f2b4985d55329ab66d82bea29d4713f6cc02.pdf