Shrinking KECCAK Hardware Implementations

Bernhard Jungk[1,2], Marc Stöttinger[3,4] and Matthias Harter[1]

1Design Informatik Medien, Hochschule RheinMain, Germany
[2]easycore GmbH, Erlangen, Germany
bernhard.jungk@easycore.com

[3]PACE Temasek Laboratories
[4]Division of Mathematical Sciences, SPMS, Nanyang Technological University, Singapore
mstottinger@ntu.edu.sg

SHA-3 Workshop 2014

Previous Evaluations

- SHA-3 evaluation focused on **high-throughput first**.
- Fast but **area-consuming** implementations.
- Later in the competition:
  - Evaluation of **lightweight** implementations.
  - Competitive hardware implementations for several platforms and trade-offs.

Previous Evaluations

- SHA-3 evaluation focused on **high-throughput first**.
- Fast but **area-consuming** implementations.
- Later in the competition:
  - Evaluation of **lightweight** implementations.
  - Competitive hardware implementations for several platforms and trade-offs.

Previous Evaluations

- SHA-3 evaluation focused on **high-throughput first**.
- Fast but **area-consuming** implementations.
- Later in the competition:
  - Evaluation of **lightweight** implementations.
  - Competitive hardware implementations for several platforms and trade-offs.

Previous Evaluations

- SHA-3 evaluation focused on **high-throughput first**.
- Fast but **area-consuming** implementations.
- Later in the competition:
  - Evaluation of **lightweight** implementations.
  - Competitive hardware implementations for several platforms and trade-offs.

Further Questions

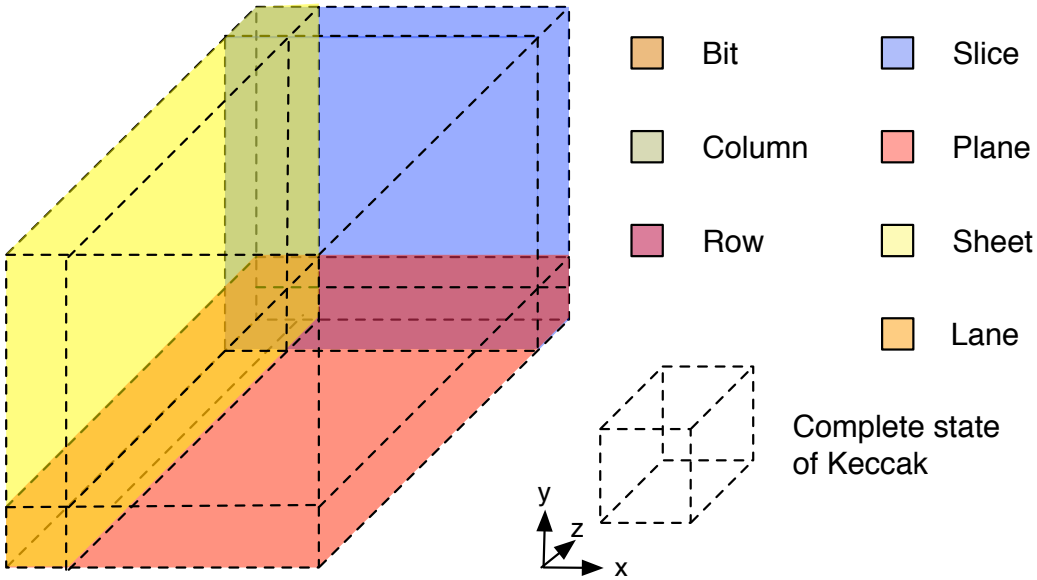
- **Question I:** Is it possible to further shrink FPGA/ASIC implementations?
- **Question II:** What is the impact of smaller state sizes on the implementations?

Further Questions

- **Question I:** Is it possible to further shrink FPGA/ASIC implementations?
- **Question II:** What is the impact of smaller state sizes on the implementations?

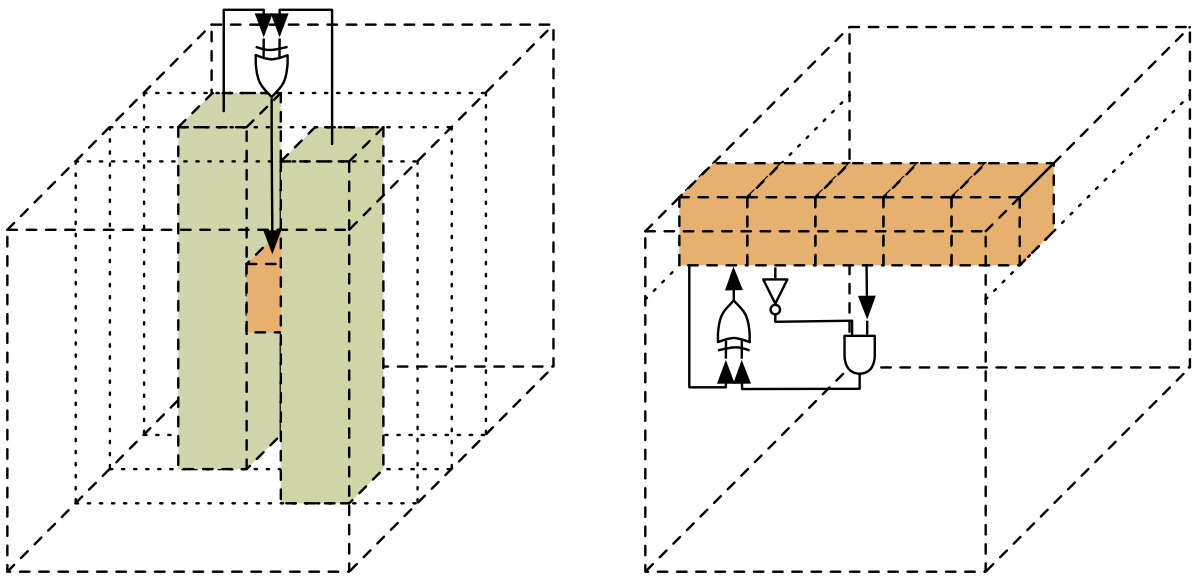
Keccak Slice-Wise Analysis

- Keccak state:



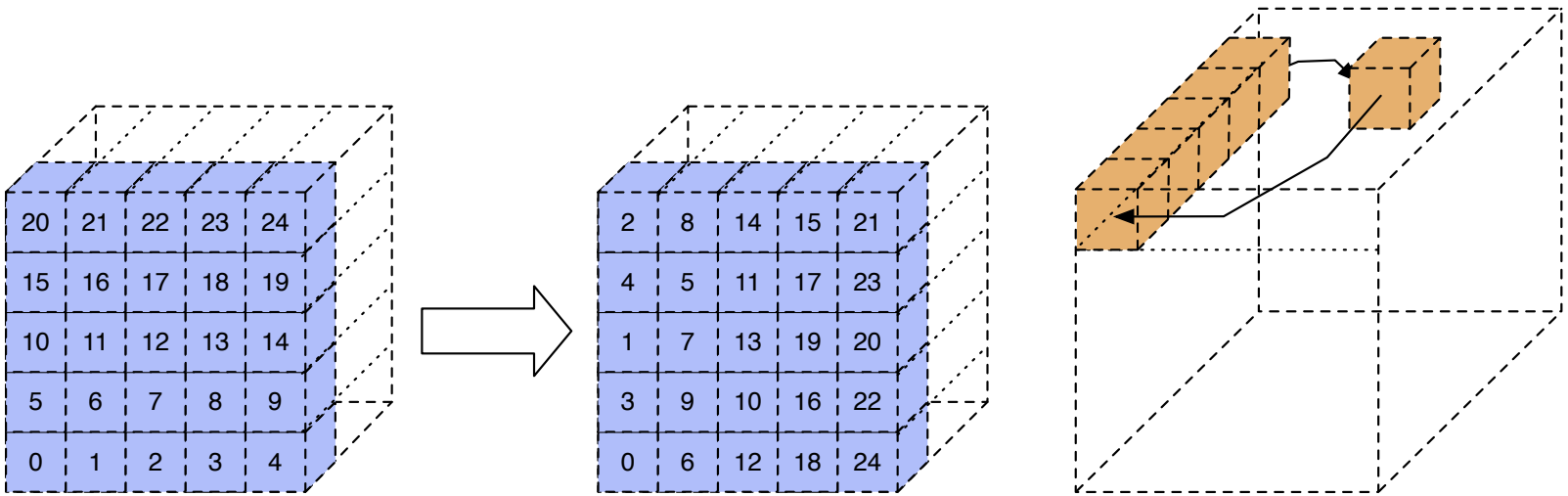
KECCAK Slice-Wise Analysis

- Relevant parts of KECCAK permutation (θ and χ).
- Introduce (foremost) intra-round dependencies.



KECCAK Slice-Wise Analysis

- Relevant parts of KECCAK round function (π and ρ).
- Introduce only inter-round dependencies.



Keccak Slice-Wise Analysis

- Slice-wise processing takes $r \cdot \frac{b}{25 \cdot n}$ clock cycles for the round function.
- b is the state size.
- r is the number of rounds.
- n is the number of slices computed in parallel.

Keccak Slice-Wise Analysis

- Slice-wise processing takes $r \cdot \frac{b}{25 \cdot n}$ clock cycles for the round function.
- b is the state size.
- r is the number of rounds.
- n is the number of slices computed in parallel.

KECCAK Slice-Wise Analysis

- Slice-wise processing takes $r \cdot \frac{b}{25 \cdot n}$ clock cycles for the round function.
- b is the state size.
- r is the number of rounds.
- n is the number of slices computed in parallel.

KECCAK Slice-Wise Analysis

- Slice-wise processing takes $r \cdot \frac{b}{25 \cdot n}$ clock cycles for the round function.
- b is the state size.
- r is the number of rounds.
- n is the number of slices computed in parallel.

Changes to the Previous Results

- Separated absorption phase from processing **(smaller but slower)**.
- Implemented parts of the processing directly as LUT-instances **(smaller)**.
- Slightly different state organization **(smaller)**.
- Additional results for lightweight variants, digest sizes (128, 160, 224, 256) and security levels ($c = 2n$ and $c = n$).
- Synthesis of a few ASIC results.

Changes to the Previous Results

- Separated absorption phase from processing **(smaller but slower)**.
- Implemented parts of the processing directly as LUT-instances **(smaller)**.
- Slightly different state organization **(smaller)**.
- Additional results for lightweight variants, digest sizes (128, 160, 224, 256) and security levels ($c = 2n$ and $c = n$).
- Synthesis of a few ASIC results.

Changes to the Previous Results

- Separated absorption phase from processing **(smaller but slower)**.
- Implemented parts of the processing directly as LUT-instances **(smaller)**.
- Slightly different state organization **(smaller)**.
- Additional results for lightweight variants, digest sizes (128, 160, 224, 256) and security levels ($c = 2n$ and $c = n$).
- Synthesis of a few ASIC results.

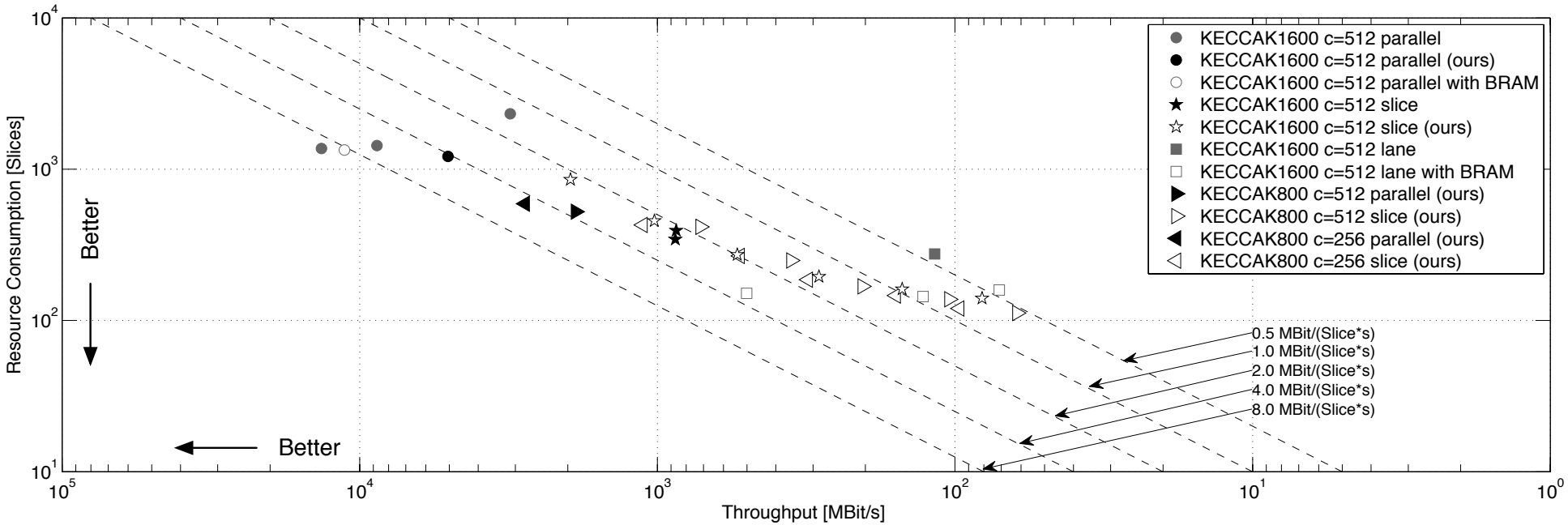
Changes to the Previous Results

- Separated absorption phase from processing **(smaller but slower)**.
- Implemented parts of the processing directly as LUT-instances **(smaller)**.
- Slightly different state organization **(smaller)**.
- Additional results for lightweight variants, digest sizes (128, 160, 224, 256) and security levels ($c = 2n$ and $c = n$).
- Synthesis of a few ASIC results.

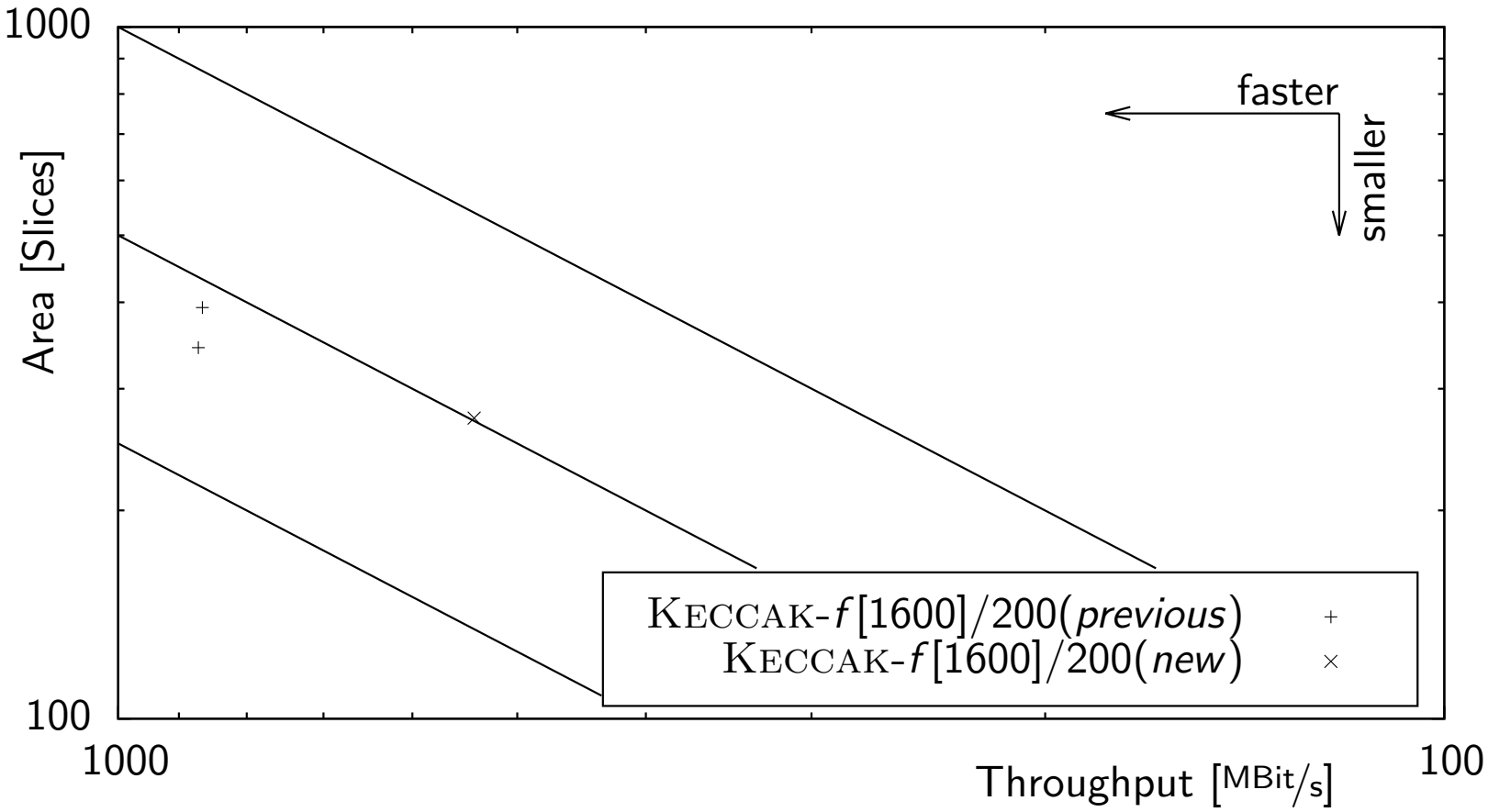
Changes to the Previous Results

- Separated absorption phase from processing **(smaller but slower)**.
- Implemented parts of the processing directly as LUT-instances **(smaller)**.
- Slightly different state organization **(smaller)**.
- Additional results for lightweight variants, digest sizes (128, 160, 224, 256) and security levels ($c = 2n$ and $c = n$).
- Synthesis of a few ASIC results.

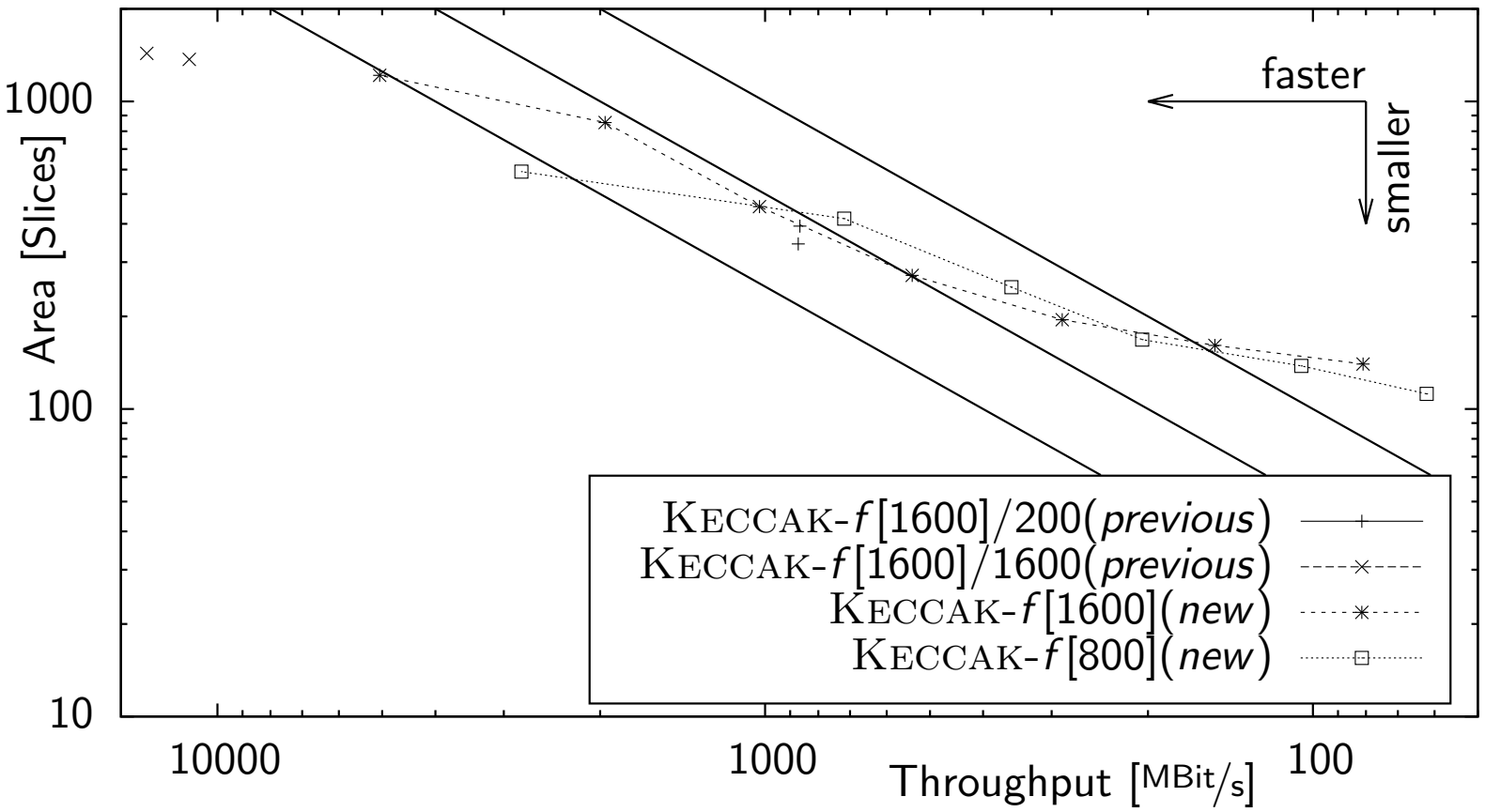
The Big Picture (KECCAK-f[1600]/[800])



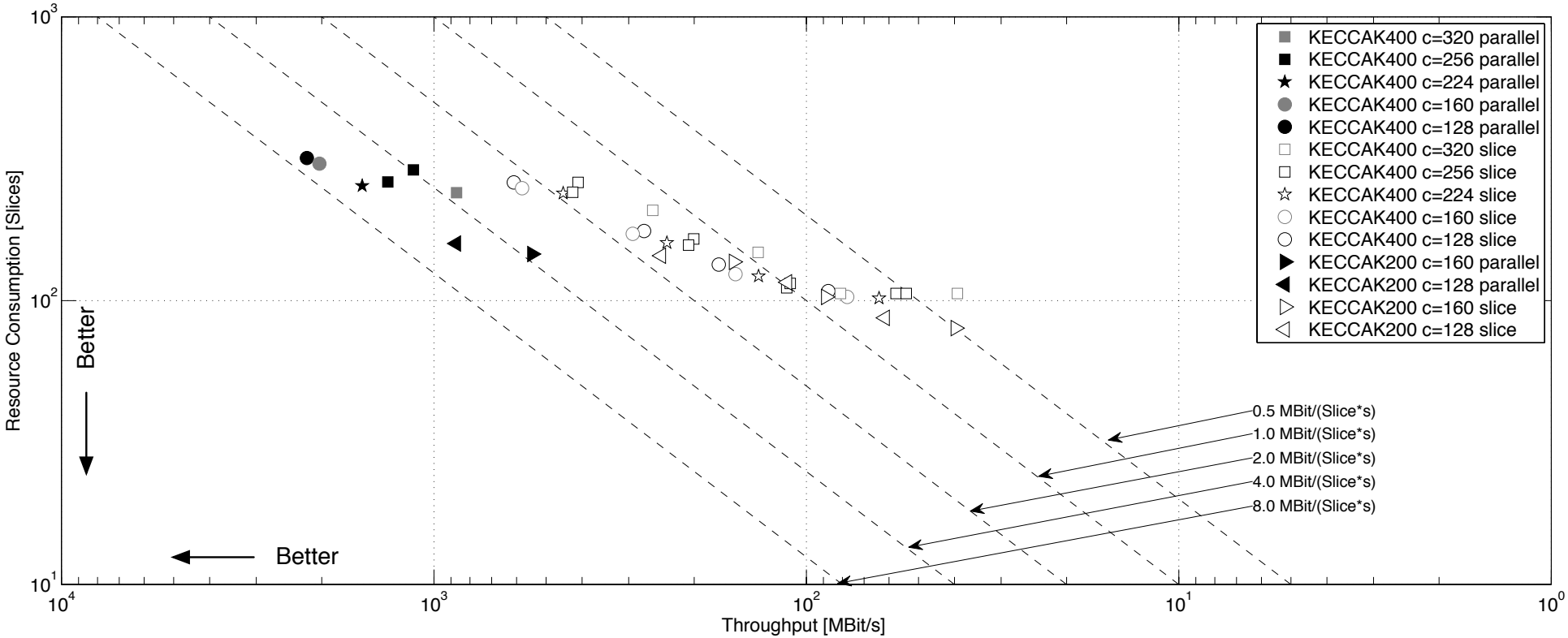
Same parameters as previous implementations



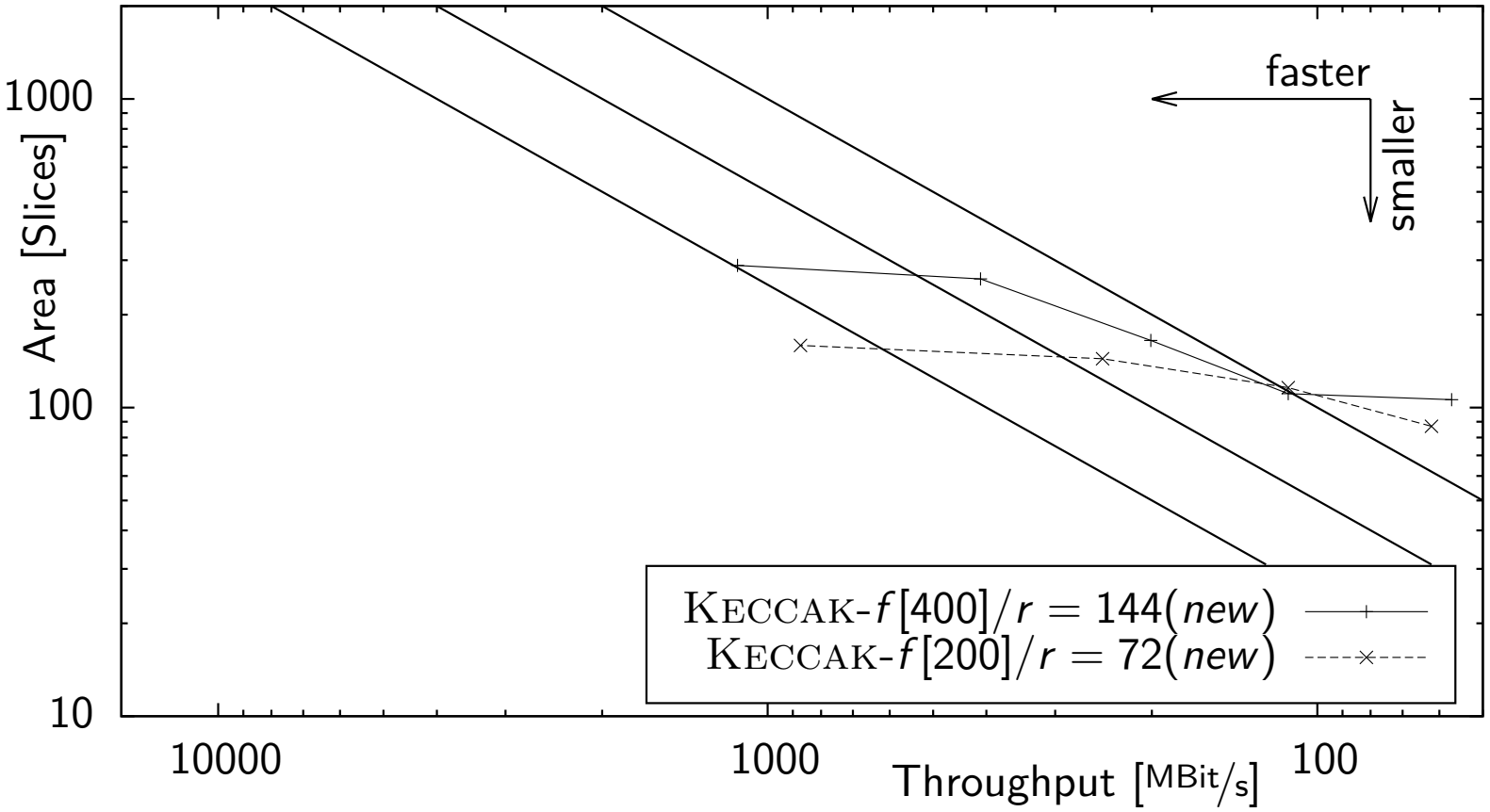
Less/More Computation in Parallel



The Big Picture (KECCAK-f[400]/[200])



Smaller state sizes



FPGA Conclusion

- KECCAK scales very well from high-throughput to lightweight applications.
- Slice-wise architectures are suitable for midrange and lightweight implementations.
- Natural barrier: Scalability limit for slice-wise implementations is the computation of **at least one slice** in parallel and a very **similar control logic**.
- Area reduction from **1600 bit** state to **200 bit** state is **only about 40%**.
- Other parallel architectures are better for high-throughput.

FPGA Conclusion

- KECCAK scales very well from high-throughput to lightweight applications.
- Slice-wise architectures are suitable for midrange and lightweight implementations.
- Natural barrier: Scalability limit for slice-wise implementations is the computation of **at least one slice** in parallel and a very **similar control logic**.
- Area reduction from **1600 bit** state to **200 bit** state is **only about 40%**.
- Other parallel architectures are better for high-throughput.

FPGA Conclusion

- KECCAK scales very well from high-throughput to lightweight applications.
- Slice-wise architectures are suitable for midrange and lightweight implementations.
- Natural barrier: Scalability limit for slice-wise implementations is the computation of **at least one slice** in parallel and a very **similar control logic**.
- Area reduction from **1600 bit** state to **200 bit** state is **only about 40%**.
- Other parallel architectures are better for high-throughput.

FPGA Conclusion

- KECCAK scales very well from high-throughput to lightweight applications.
- Slice-wise architectures are suitable for midrange and lightweight implementations.
- Natural barrier: Scalability limit for slice-wise implementations is the computation of **at least one slice** in parallel and a very **similar control logic**.
- Area reduction from **1600 bit** state to **200 bit** state is **only about 40%**.
- Other parallel architectures are better for high-throughput.

FPGA Conclusion

- KECCAK scales very well from high-throughput to lightweight applications.
- Slice-wise architectures are suitable for midrange and lightweight implementations.
- Natural barrier: Scalability limit for slice-wise implementations is the computation of **at least one slice** in parallel and a very **similar control logic**.
- Area reduction from **1600 bit** state to **200 bit** state is **only about 40%**.
- Other parallel architectures are better for high-throughput.

ASIC Results for KECCAK-f[400] and KECCAK-f[200].

Variant	Capacity (c) [Bit]	Rate (r) [Bit]	Architecture	Process	Area kGE	Frequency [Mhz]	Throughput [MBit/s]	Throughput @ 100 kHz [MBit/s]
Photon-256/32/32	256	32	[17]	UMC 65nm	2.17	N/A	N/A	0.003
Photon-160/36/36	160	36	[17]	UMC 65nm	1.39	N/A	N/A	0.0027
Photon-128/16/16	128	16	[17]	UMC 65nm	1.12	N/A	N/A	0.0016
KECCAK-f[1600]	512	1088	[14]	UMC 130nm	5.52	N/A	N/A	0.0044
KECCAK-f[400]	256	144	This paper	UMC 65nm	13.84	860	258	0.03
			This paper	UMC 65nm	8.69	477	143	0.03
			This paper	AMS 350nm	7.51	178	53.4	0.03
			This paper	AMS 350nm	6.64	104	31.2	0.03
			[13]	130nm	5.09	N/A	N/A	0.0144
	160	240	This paper	UMC 65nm	13.93	842	350	0.04
			This paper	UMC 65nm	8.80	457	190	0.04
			This paper	AMS 350nm	7.60	176	73.3	0.04
			This paper	AMS 350nm	6.68	109	45.4	0.04
KECCAK-f[200]	128	72	This paper	UMC 65nm	8.63	908	291	0.032
			This paper	UMC 65nm	5.09	482	154	0.032
			This paper	AMS 350nm	4.50	178	57.2	0.032
			This paper	AMS 350nm	3.87	113	36.3	0.032
			[13]	130nm	2.52	N/A	N/A	0.008

ASIC Conclusion

- ASIC evaluation inconclusive (more research needed).
- Our results are larger than published ones [13], but faster.
- Previous results [14] show that the full KECCAK can be implemented with less GE than all our 400 bit variants.
- Assumption: KECCAK could be competitive to other lightweight hash functions with an architecture similar to [14], which is tailored towards ASICs.

ASIC Conclusion

- ASIC evaluation inconclusive (more research needed).
- Our results are larger than published ones [13], but faster.
- Previous results [14] show that the full KECCAK can be implemented with less GE than all our 400 bit variants.
- Assumption: KECCAK could be competitive to other lightweight hash functions with an architecture similar to [14], which is tailored towards ASICs.

ASIC Conclusion

- ASIC evaluation inconclusive (more research needed).
- Our results are larger than published ones [13], but faster.
- Previous results [14] show that the full KECCAK can be implemented with less GE than all our 400 bit variants.
- Assumption: KECCAK could be competitive to other lightweight hash functions with an architecture similar to [14], which is tailored towards ASICs.

ASIC Conclusion

- ASIC evaluation inconclusive (more research needed).
- Our results are larger than published ones [13], but faster.
- Previous results [14] show that the full KECCAK can be implemented with less GE than all our 400 bit variants.
- Assumption: KECCAK could be competitive to other lightweight hash functions with an architecture similar to [14], which is tailored towards ASICs.

Questions?