Improved Montgomery Multiplication

Trenton J. Grale
Earl E. Swartzlander, Jr.

ARITH 2023

Montgomery Multiplication Basics

- Operands in "Montgomery Domain"
- Montgomery product $P = ABR^{-1} \bmod M$
- Computation:
  - $T = AB$ $\quad$ $T_0 = T \bmod R$
  - $Q = T_0M' \bmod R$ $\quad$ $Q_0 = Q \bmod M$
  - $U = Q_0M$
  - $P = (T + U) / R$
  - If ($P > M$): $P = P - M$
- Can be performed at digit or bit level

Serial Montgomery Model

Reduction Mode		Digit Scanning Priority		
		Operand	Hybrid	Product
Separated		SOS		
Integrated	Coarse	CIOS	CIHS	
	Fine	FIOS		FIPS

n Operand word size (bits)

d Digit size (bits)

k Number of digits = $\lceil n/d \rceil$

Category	Scanning Order	# Cycles
SOS	k^2, $k(1, k)$	$2k^2 + k$
CIOS	$k(k, 1, k)$	$2k^2 + k$
FIOS	$k[1, 1, 1, 2(k-1)]$	$2k^2 + k$

Serial Montgomery Implementations

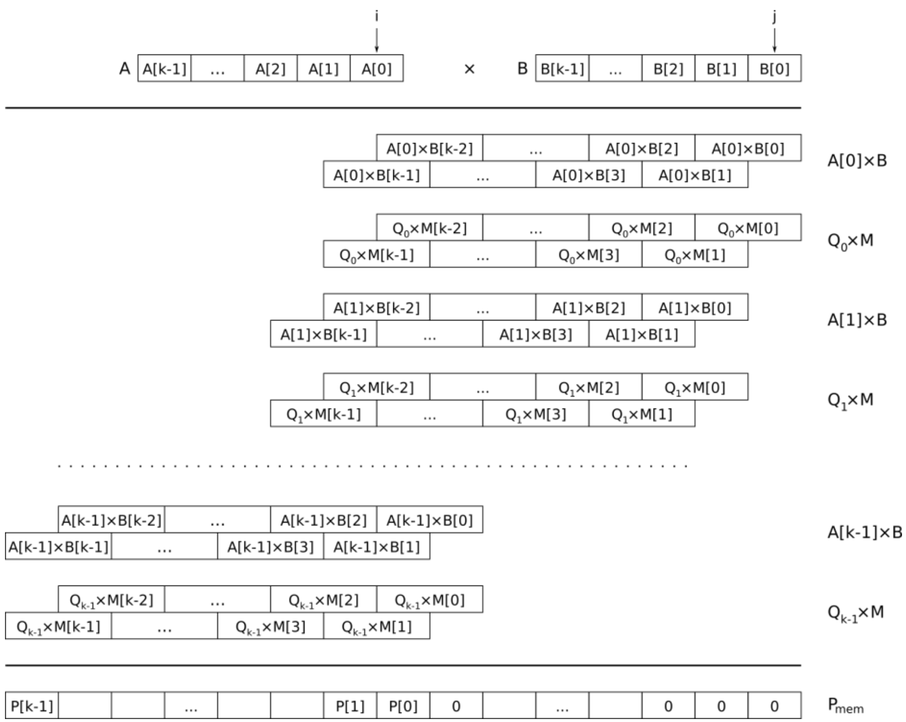
- Eberle, *et al.* digit-digit architecture
- Großschädl, *et al.* bit-word architecture
- Tenca & Koç bit-digit architecture

Architecture	Base Operand	# Base Operations	# Cycles	Koç Classification
Eberle	Digit	$2k^2 + k$	$2k^2 + k$	CIOS
Großschädl	Bit/word	n	$n + k$	FIOS[a]
Tenca & Koç	Bit/digit	nk	$2n + k - 1$	FIOS[a]

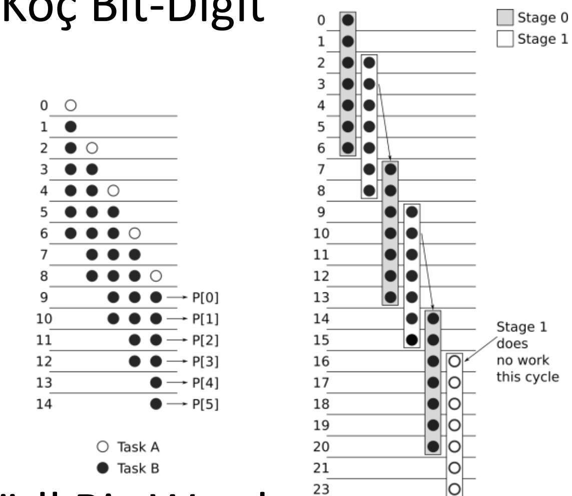
[a]Closest fit

Serial Architectures

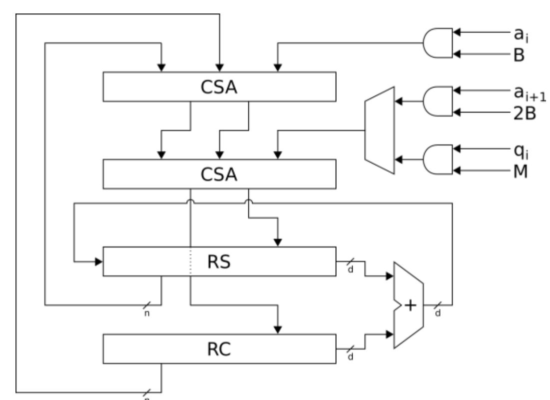
Eberle Digit-Digit



Tenca & Koç Bit-Digit



Großschädl Bit-Word



Extended Serial Montgomery Model

- Other scanning and reductions modes are possible
- Separated Product Scanning (SPS)
- Digit level parallelism—schedule multiple concurrent operand or product digit computations
- m: Number of digit multipliers

Extended Serial Montgomery Model

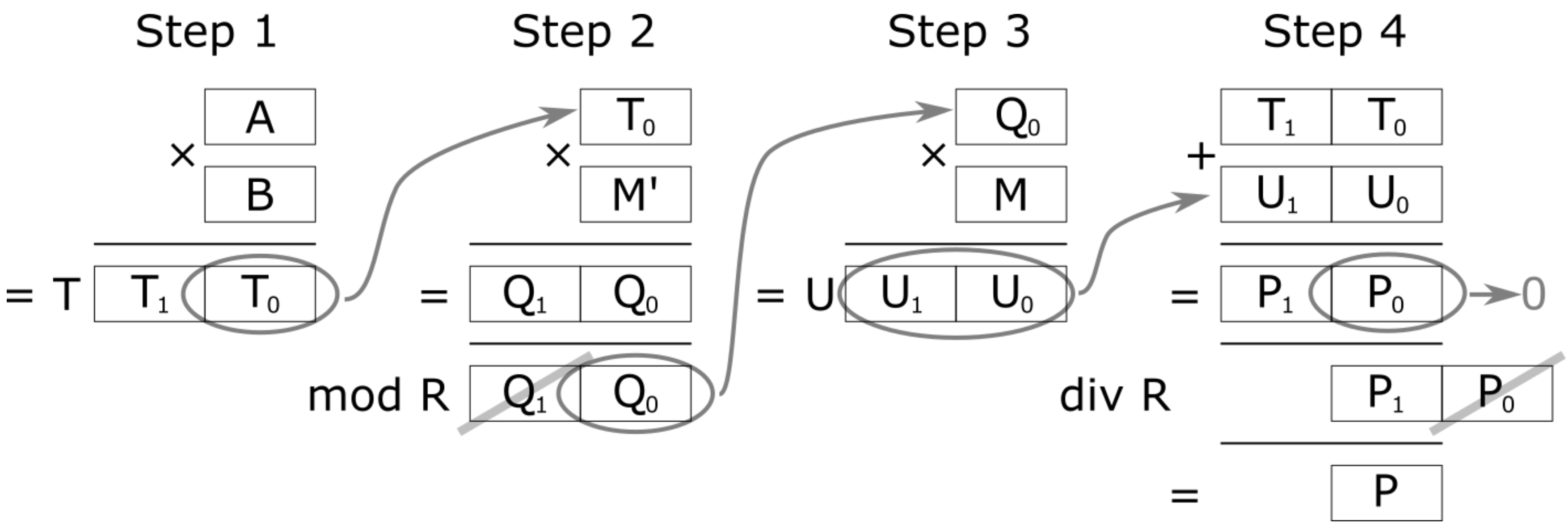
Reduction Mode		Digit Scheduling Priority		
		Operand	Hybrid	Product
Separated		SOS/m		**SPS/m**
Integrated	Coarse	CIOS/m	CIHS/m	
	Fine	FIOS/m		FIPS/m

Category	Schedule Order ($m = 1$)	# Cycles	Schedule Order ($m > 1$)	# Cycles
SOS	$k^2, k(1, k)$	$2k^2 + k$	$\lceil k^2/m \rceil, k(1, \lceil k/m \rceil)$	$\lceil k^2/m \rceil + k\lceil k/m \rceil + k$
CIOS	$k(k, 1, k)$	$2k^2 + k$	$k(\lceil k/m \rceil, 1, \lceil k/m \rceil)$	$2k\lceil k/m \rceil + k$
FIOS	$k[1, 1, 1, 2(k-1)]$	$2k^2 + k$	$k[1, 1, 1, \lceil 2(k-1)/m \rceil]$	$k\lceil 2(k-1)/m \rceil + 3k$
SPS	$k^2, (k^2 + k)/2, k^2$	$2.5k^2 + 0.5k$	$\lceil [k^2, (k^2+k)/2, k^2]/m \rceil$	$\lceil (2.5k^2 + 0.5k)/m \rceil$

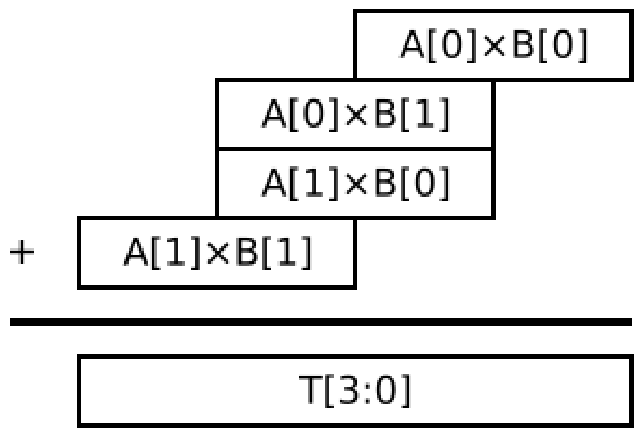
Cycle Counts with Digit Level Parallelism (k = 4)

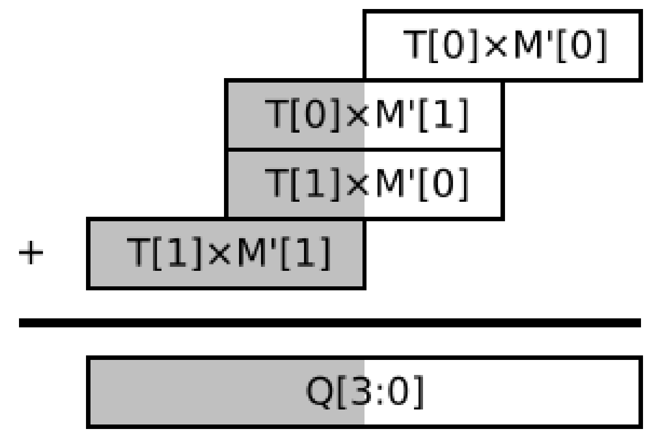
k	*m*	SOS	CIOS	FIOS	SPS
4	1	36	36	36	42
	2	20	20	24	21
	3	18	20	20	14
	4	12	12	20	11
	5	12	12	20	9

Montgomery Macro Optimization



Digit Multiplication, k = 2





$N_P = k^2$

$N_Q = (k^2 + k) / 2$

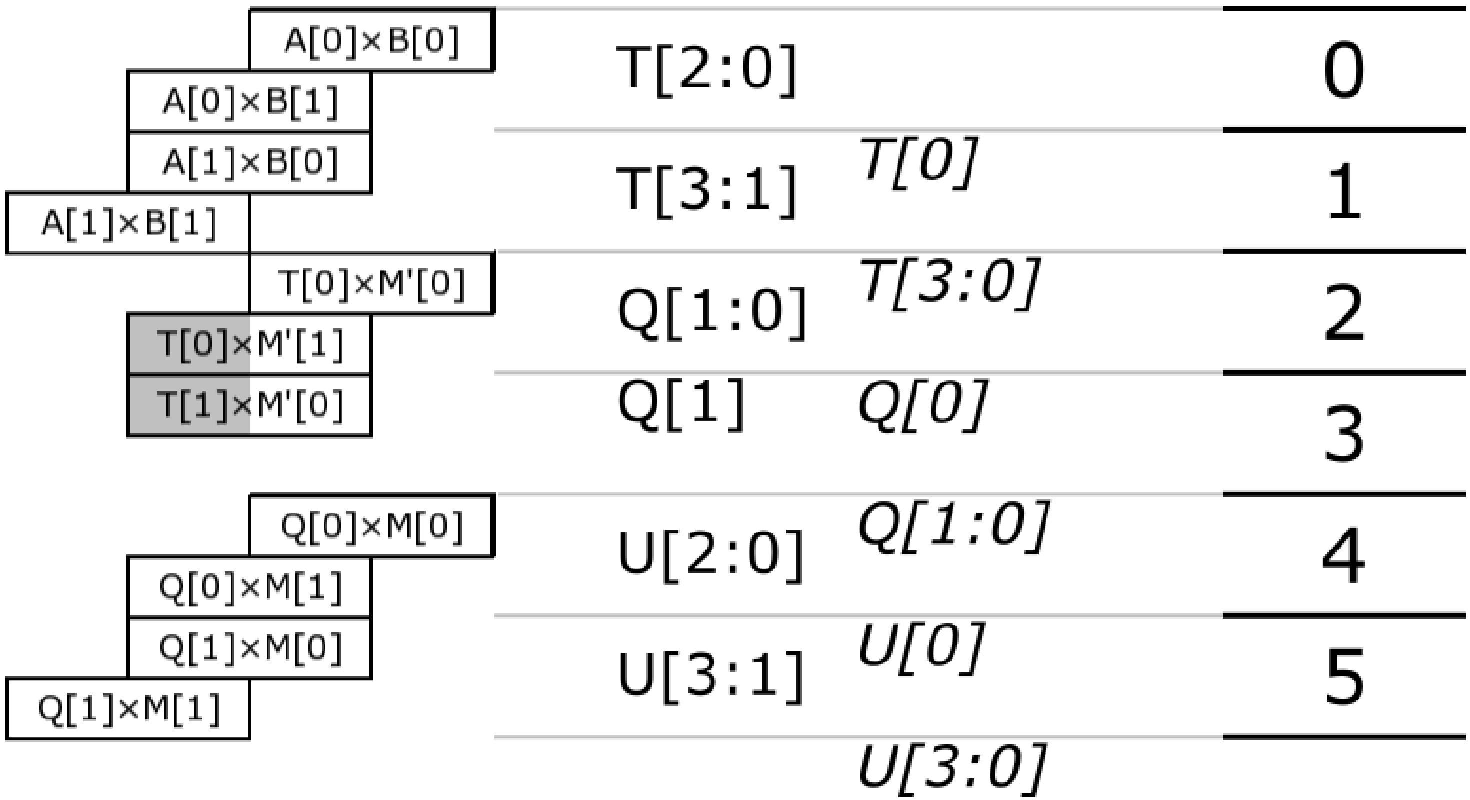
$\Rightarrow 0.5N_P$ for large k

Rescheduled Montgomery Multipliers (RMM)

- Digit multiplication for granularity
- Opportunistically defer T_1 computations
- Avoid unnecessary computations: Q_0 only
- Final sum $T_1 + U_1$ + ones_detect(T_0)
- Multiple digit products in parallel
- Vertically-biased accumulation to minimize carry propagation

RMM (2, 1) Schedule

RMM (2, 2) Schedule



RMM (2, 1) and (2, 2) Pipelines

Required:					*T[0]*	*T[0]*	*T[1]*	*Q[0]*	*Q[0]*	*Q[1]*	*Q[1]*			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
L	T[1:0]	T[2:1]	T[2:1]	T[3:2]	Q[1:0]	Q[2:1]	Q[2:1]	U[1:0]	U[2:1]	U[2:1]	U[3:2]			
M		T[1:0]	T[2:1]	T[2:1]	T[3:2]	Q[1:0]	Q[2:1]	Q[2:1]	U[1:0]	U[2:1]	U[2:1]	U[3:2]		
A			T[1:0]	T[2:1]	T[2:1]	T[3:2]	Q[1:0]	Q[2:1]	Q[2:1]	U[1:0]	U[2:1]	U[2:1]	U[3:2]	
			0	1	2	3	4	5	6	7	8	9	10	11
Ready:				*T[0]*		*T[1:0]*	*T[3:0]*	*Q[0]*		*Q[1:0]*	*U[0]*		*U[1:0]*	*U[3:0]*

Required:			*T[0]*	*T[1]*	*Q[0]*	*Q[1]*								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
L	T[2:0]	T[3:1]	Q[1:0]	Q[1]	U[2:0]	U[3:1]								
M		T[2:0]	T[3:1]	Q[1:0]	Q[1]	U[2:0]	U[3:1]							
A			T[2:0]	T[3:1]	Q[1:0]	Q[1]	U[2:0]	U[3:1]						
			0	1	2	3	4	5	6	7	8	9	10	11
Ready:				*T[0]*	*T[3:0]*	*Q[0]*	*Q[1:0]*	*U[0]*	*U[3:0]*					

Comparisons

- Previous serial Montgomery architectures
  - Eberle digit-digit serial
  - Großschädl bit-word serial
  - Tenca bit-digit serial
- Rescheduled Montgomery Multiplier
- Other approaches
  - Basic synthesized multipliers
  - Full directly-realized Montgomery designs
  - McIvor full-word pipelined multiplier and ECC Processor

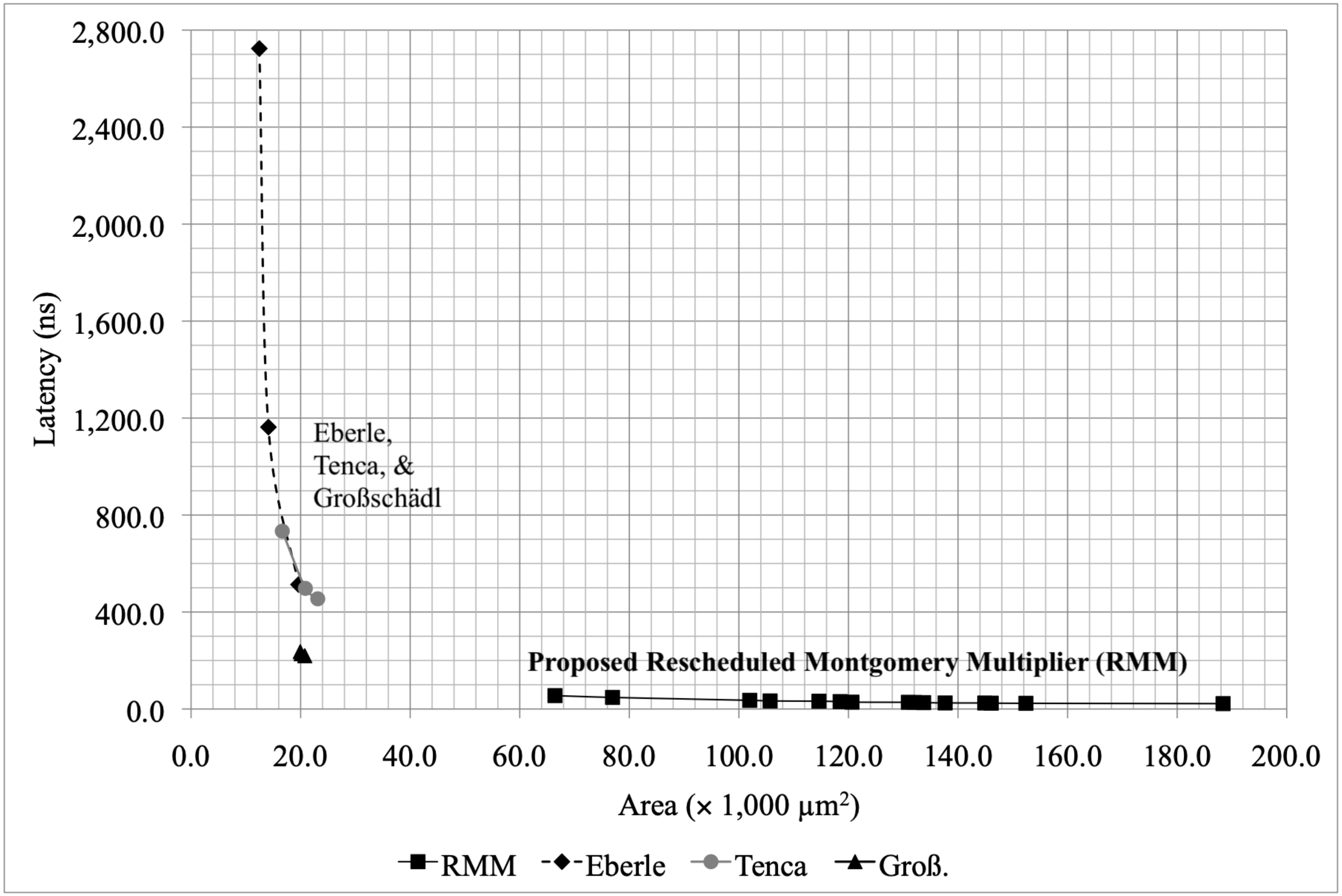
Serial Montgomery Multiplier Results: Eberle, Großschädl, Tenca & Koç

Design	Area (k μm²)	Latency (ns)	A·L Product
Eberle digit-digit, *d* = 8	12.5	2,724.0	33.95
Eberle digit-digit, *d* = 16	14.1	1,162.5	16.42
Eberle digit-digit, *d* = 32	19.6	513.3	10.06
Großschädl bit-word, *d* = 8	19.9	236.5	4.72
Großschädl bit-word, *d* = 16	20.1	227.7	4.58
Großschädl bit-word, *d* = 32	20.8	219.8	4.56
Tenca & Koç bit-digit, *d* = 8	21.3	1,169.5	24.95
Tenca & Koç bit-digit, *d* = 16	18.5	1,294.5	23.89
Tenca & Koç bit-digit, *d* = 32	18.7	1,485.9	27.72

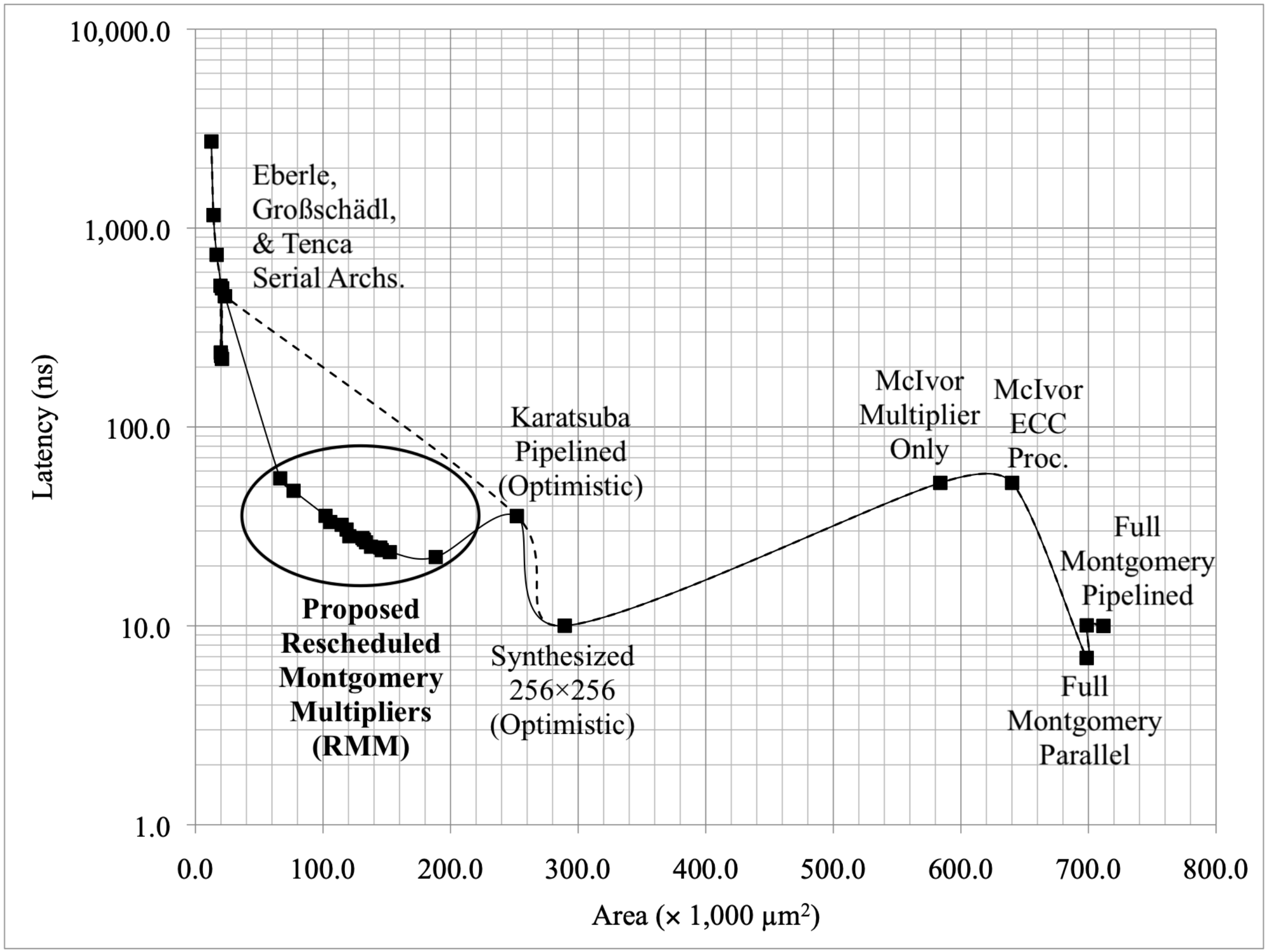
Rescheduled Montgomery Multiplier Builds and Results

# Digits *k*	# Bits/Digit *d*	# Digit Multipliers *m*	Area Range (k μm^2)	Latency Range (ns)	A·L Range
2	128	<u>1</u>, 2	106 – 188	22.2 – 33.3	**3.52** – 4.18
3	86	1, 2, <u>3</u>	66 – 146	24.0 – 55.1	**3.51** – 3.69
4	64	2, 3, <u>4</u>, 5	77 – 145	24.8 – 47.8	**3.40** – 3.67
5	52	4, <u>5</u>, 6	116 – 163	29.1 – 37.2	**3.61** – 4.33
6	43	5, 6, 7, 8, <u>9</u>, 10	110 – 152	23.5 – 40.2	**3.44** – 4.44
7	37	6, 7, 8, 9, <u>10</u>, 11	113 – 142	25.6 – 41.2	**3.62** – 4.72
8	32	8, 9, 10, 11, 12, <u>13</u>, 14	114 – 142	26.3 – 38.8	**3.52** – 4.42

Latency versus Area: Serial and RMM



Overall Latency versus Area



Conclusions

- First order estimate with “standard” multipliers of various sizes is idealized
- Only full direct parallel and pipelined architectures are faster, at high die area cost
- RMMs have better performance than McIvor in only 25% (or less) area
- RMM max size 7× serialized architectures but one to two *orders of magnitude* better latency
- RMMs do not do repeated bit or digit Montgomery reductions—reduction saved for the end

Conclusions

- RMMs best A·L product of any practical Montgomery multipliers that were implemented
- RMM (4, 4) versus (2, 1) incurs 14% area cost for speedup = 1.18
  - Reduced Q_0 computation helps
- Montgomery optimizations and overlapping products provide small but significant performance benefits