Institute of Microelectronic Systems

Leibniz Universität Hannover

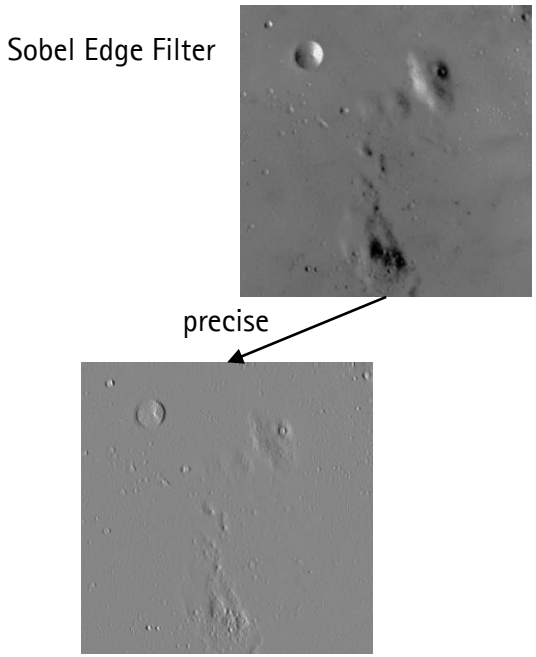
ATE-Accuracy Trade-Offs for Approximate Adders and Multipliers in Pipelined Processor Datapaths

M. Weißbrich[1], A. Najafi[2], A. García-Ortiz[2] and G. Payá-Vayá[1]

1) Institute of Microelectronic Systems, Leibniz Universität Hannover
2) Institute of Electrodynamics and Microelectronics, Universität Bremen

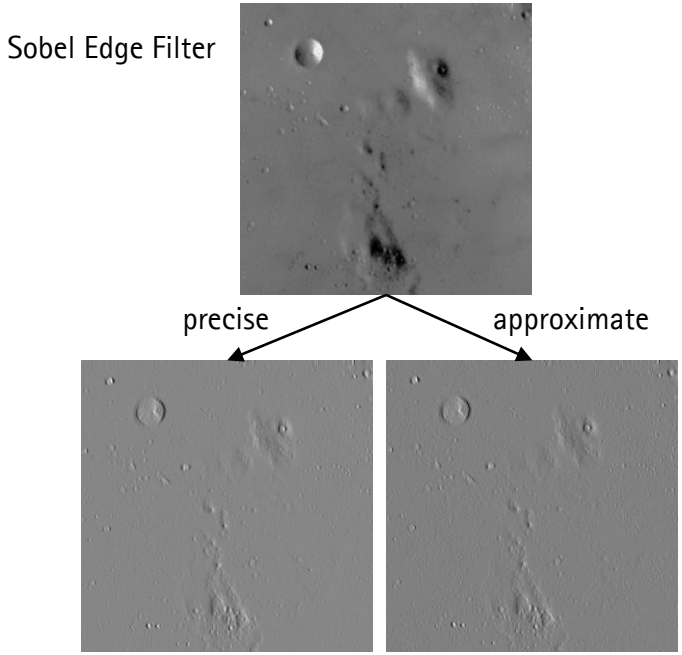
Introduction: Approximate Computing

- Precise computation results not necessary in many applications
  - (Noisy) image processing, edge & feature detection etc.
- Imprecise results in hardware by…



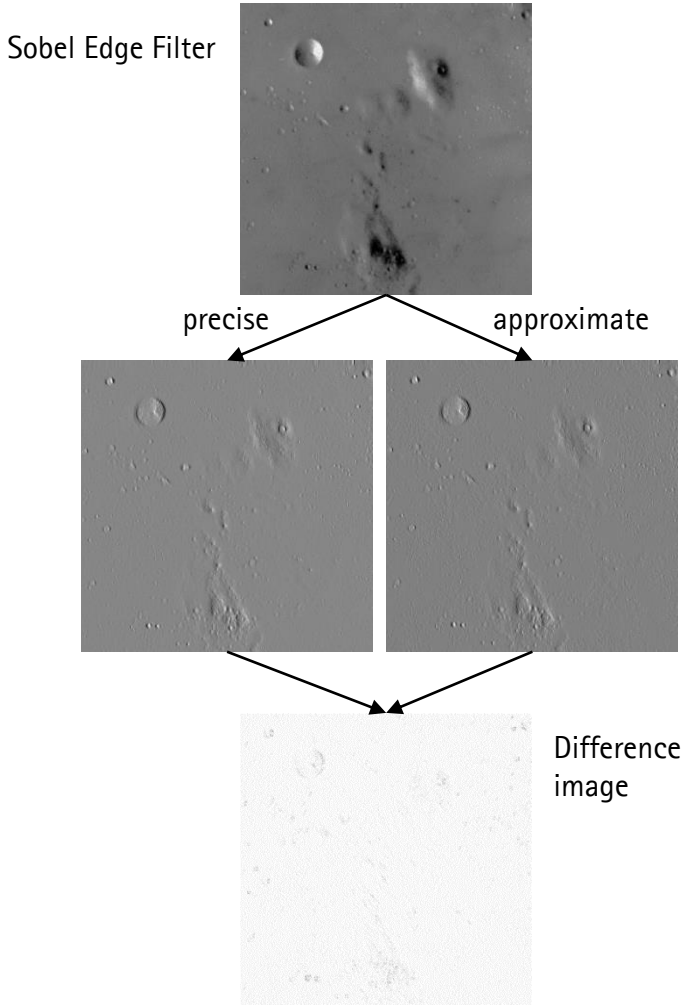
Introduction: Approximate Computing

- Precise computation results not necessary in many applications
  - (Noisy) image processing, edge & feature detection etc.
- Imprecise results in hardware by…
  - *Approximate Computing*: Deterministic designs without timing violations



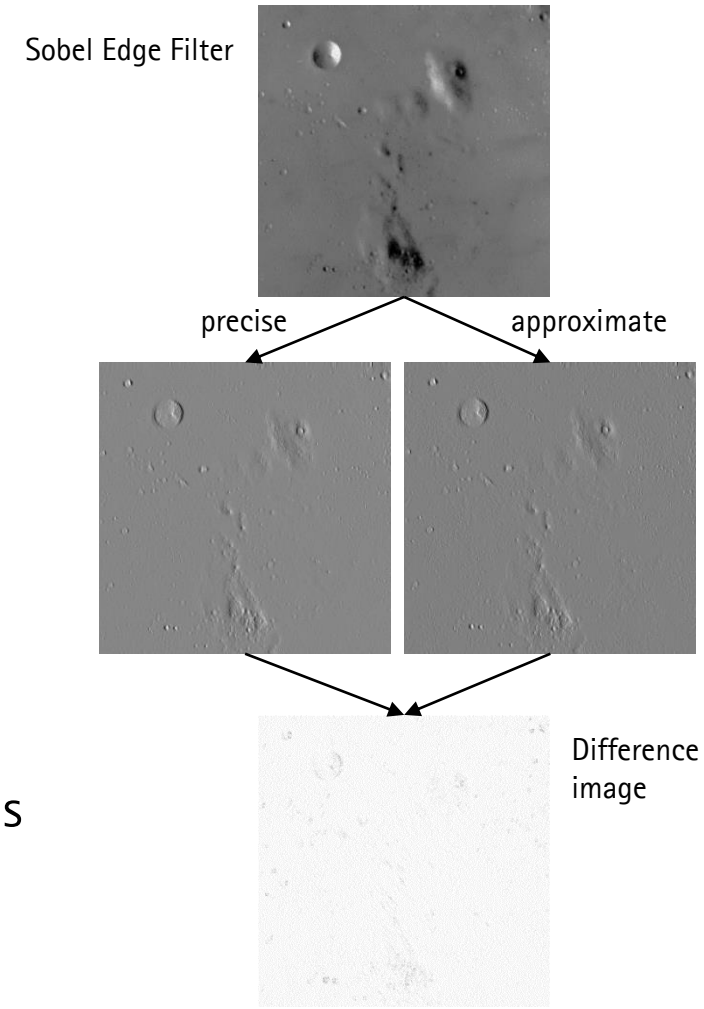
Introduction: Approximate Computing

- Precise computation results not necessary in many applications
  - (Noisy) image processing, edge & feature detection etc.
- Imprecise results in hardware by…
  - *Approximate Computing*: Deterministic designs without timing violations



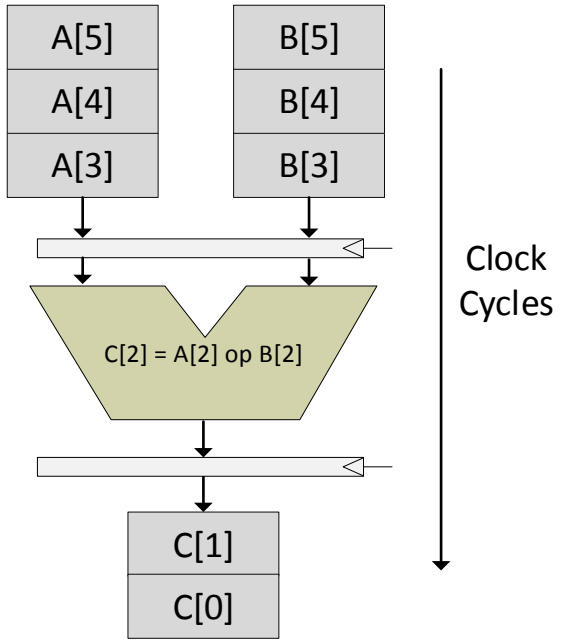
Introduction: Approximate Computing

- Precise computation results not necessary in many applications
  - (Noisy) image processing, edge & feature detection etc.
- Imprecise results in hardware by…
  - *Approximate Computing*:
    Deterministic designs without timing violations
  - *Stochastic Computing*:
    Deterministic designs with timing violations
    - Not considered here



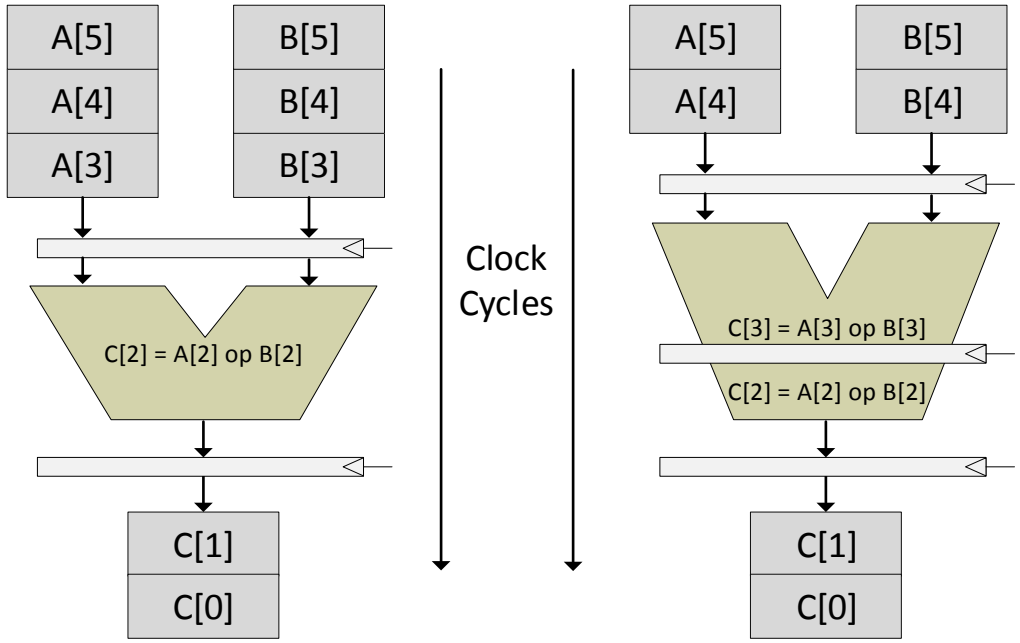
Motivation: Pipelined Approximate Units

- Pipelining of processor architectures for image processing:
  - Independent data, no pipeline conflicts



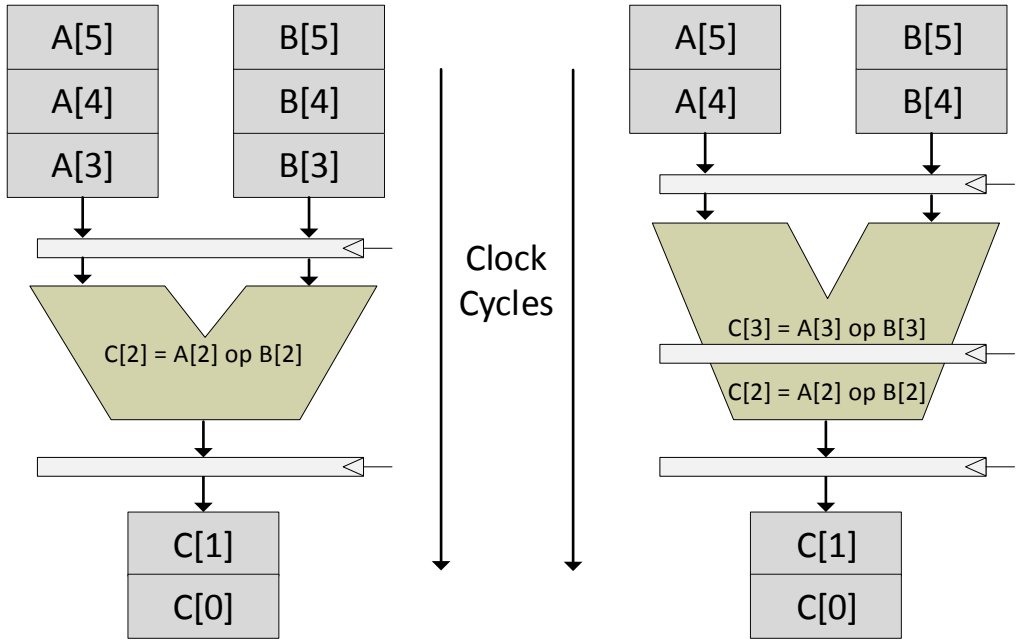
Motivation: Pipelined Approximate Units

- Pipelining of processor architectures for image processing:
  - Independent data, no pipeline conflicts
  - Direct performance boost for long vector operations



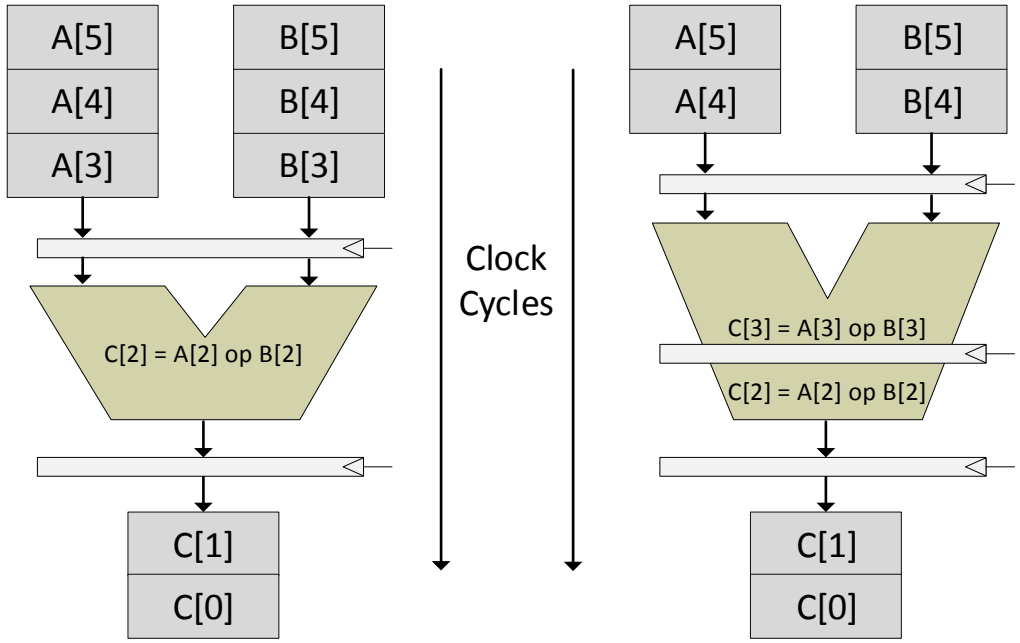
Motivation: Pipelined Approximate Units

- Pipelining of processor architectures for image processing:
  - Independent data, no pipeline conflicts
  - Direct performance boost for long vector operations
  - Pipelining not considered in comparisons of approximate arithmetic



Motivation: Pipelined Approximate Units

- Pipelining of processor architectures for image processing:
  - Independent data, no pipeline conflicts
  - Direct performance boost for long vector operations
  - Pipelining not considered in comparisons of approximate arithmetic
- Influence on:
  - Area Efficiency
  - Energy Efficiency
  - Needs to be explored for architectural decisions



Implementation & Flow Strategy

1. Generic VHDL description
   - Precise arithmetic described behaviorally
   - Tool selects efficient implementation considering user constraints

Implementation & Flow Strategy

1. Generic VHDL description
   - Precise arithmetic described behaviorally
   - Tool selects efficient implementation considering user constraints

VHDL

```
C <= A + B;
```

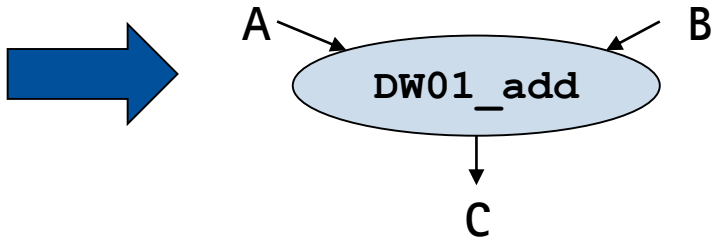
Implementation & Flow Strategy

1. Generic VHDL description
   - Precise arithmetic described behaviorally
   - Tool selects efficient implementation considering user constraints

VHDL

```
C <= A + B;
```

Inference by Synopsys Design Compiler / DesignWare Library



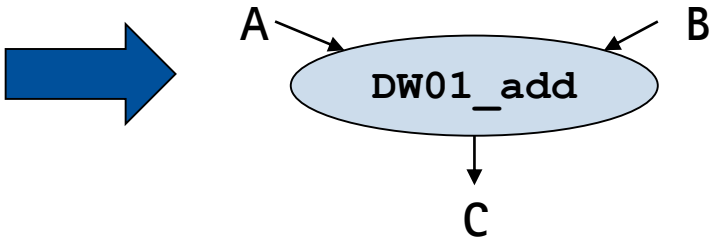
Implementation & Flow Strategy

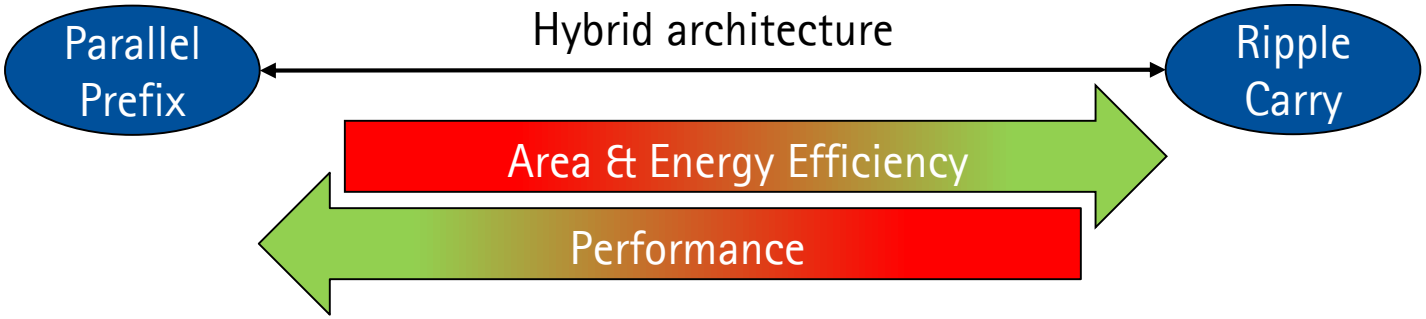
1. Generic VHDL description
   - Precise arithmetic described behaviorally
   - Tool selects efficient implementation considering user constraints

VHDL

```
C <= A + B;
```

Inference by Synopsys Design Compiler / DesignWare Library





Implementation & Flow Strategy

2. Pipeline
   - Described behaviorally by output registration
   - Tool performs retiming/register balancing

Implementation & Flow Strategy

2. Pipeline
   - Described behaviorally by output registration
   - Tool performs retiming/register balancing

VHDL

```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Implementation & Flow Strategy

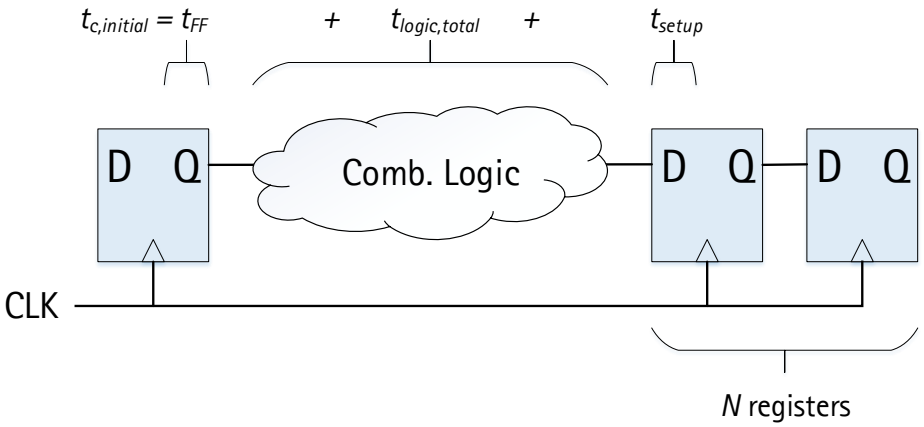
2. Pipeline

- Described behaviorally by output registration
- Tool performs retiming/register balancing

VHDL

```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Synthesis



Implementation & Flow Strategy

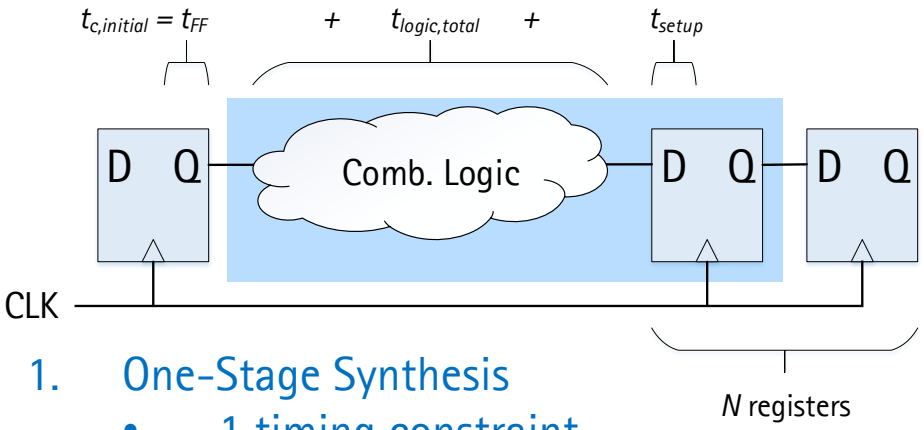
2. Pipeline

- Described behaviorally by output registration
- Tool performs retiming/register balancing

VHDL

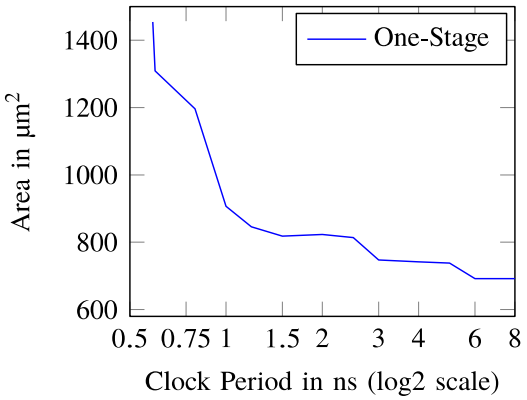
```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Synthesis



1. One-Stage Synthesis
   - 1 timing constraint

Implementation & Flow Strategy

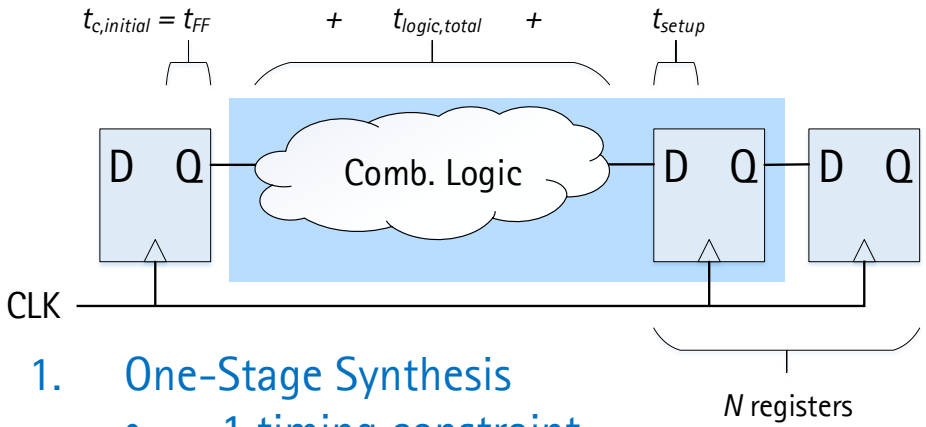


2. Pipeline
   - Described behaviorally by output registration
   - Tool performs retiming/register balancing

VHDL

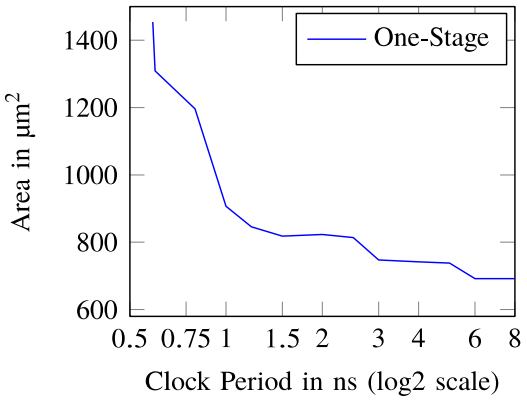
```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Synthesis



1. One-Stage Synthesis
   - 1 timing constraint

Implementation & Flow Strategy

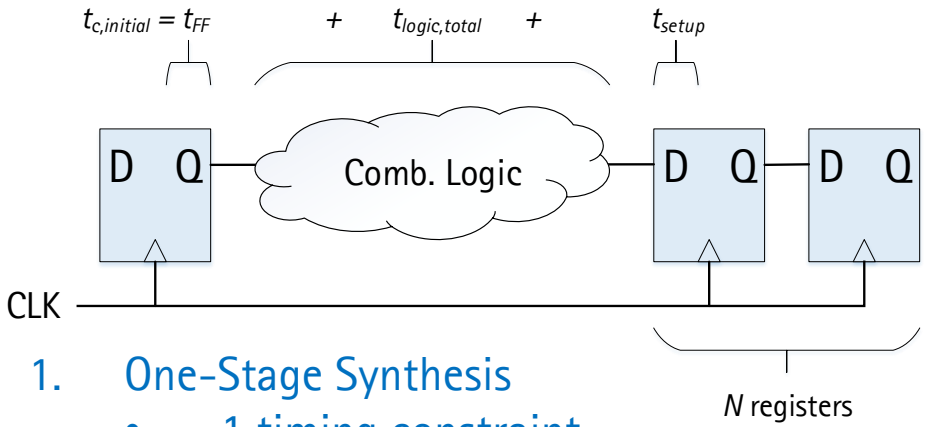


2. Pipeline
   - Described behaviorally by output registration
   - Tool performs retiming/register balancing

VHDL

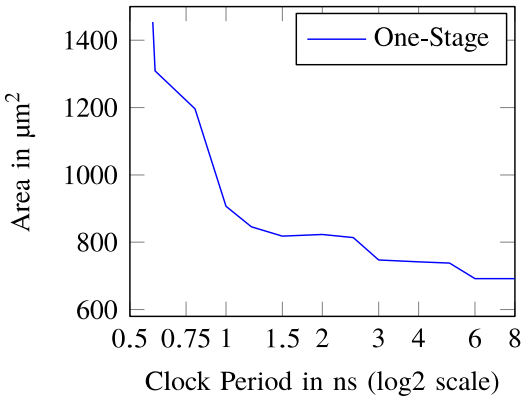
```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Synthesis



1. One-Stage Synthesis
   - 1 timing constraint

Implementation & Flow Strategy

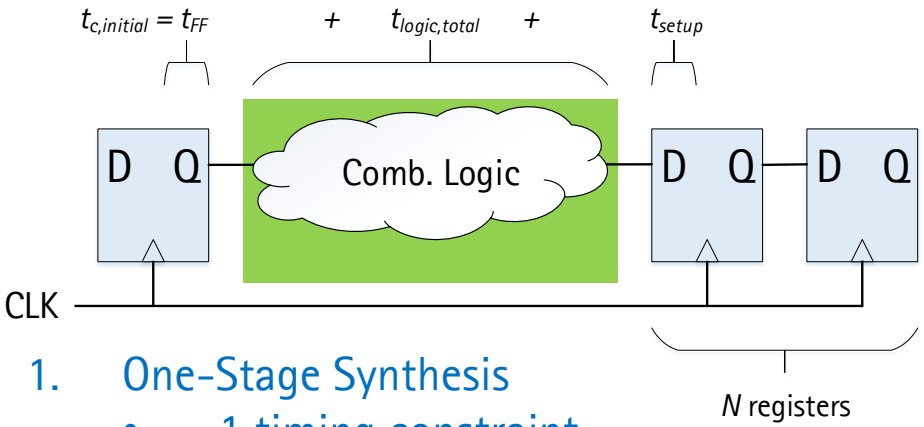


2. Pipeline
   - Described behaviorally by output registration
   - Tool performs retiming/register balancing

VHDL

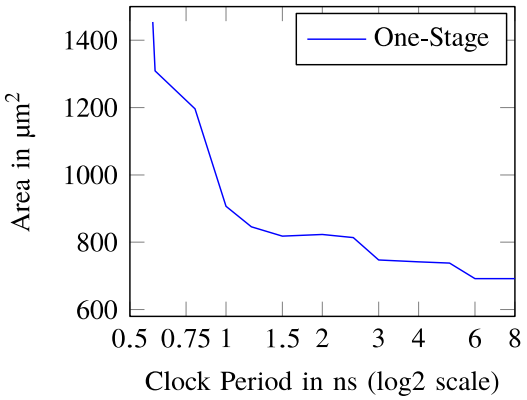
```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Synthesis



1. One-Stage Synthesis
   - 1 timing constraint
2. Two-Stage Synthesis
   - *Relaxed* constraint for selection

Implementation & Flow Strategy

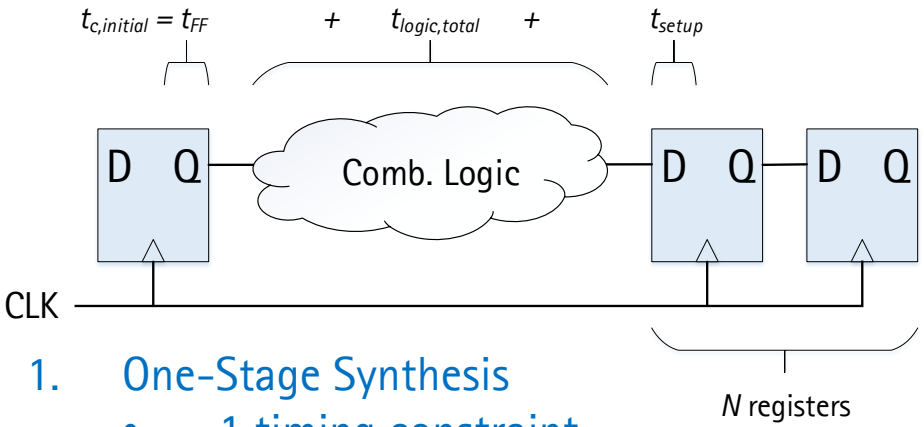


2. Pipeline
   - Described behaviorally by output registration
   - Tool performs retiming/register balancing

VHDL

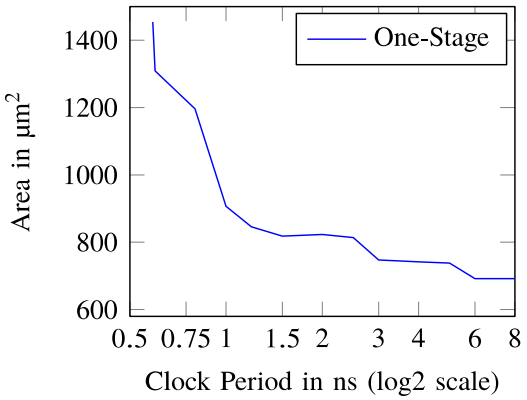
```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Synthesis



1. One-Stage Synthesis
   - 1 timing constraint
2. Two-Stage Synthesis
   - *Relaxed* constraint for selection

Implementation & Flow Strategy

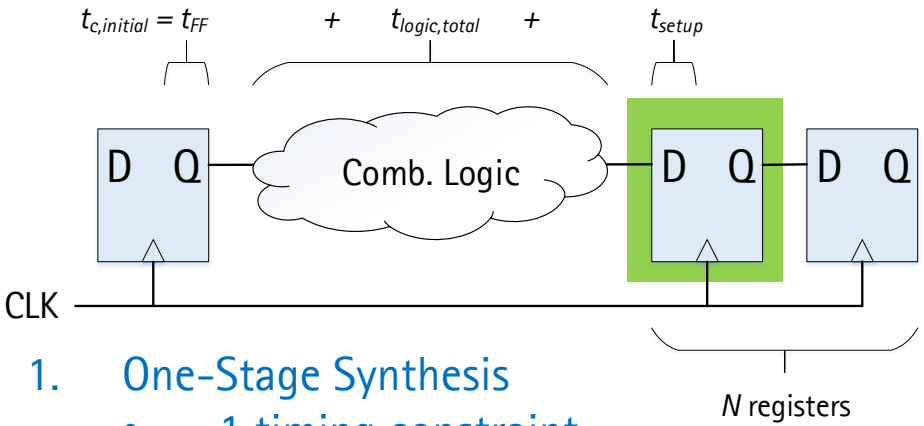


2. Pipeline
   - Described behaviorally by output registration
   - Tool performs retiming/register balancing

VHDL

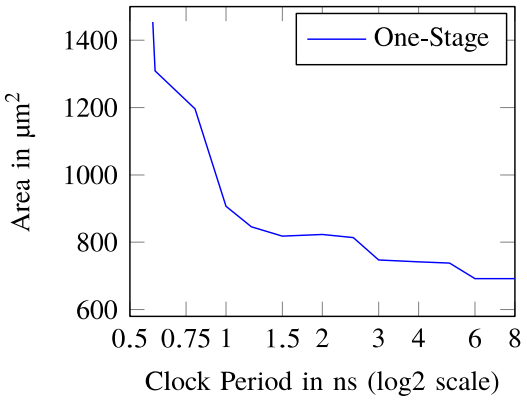
```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Synthesis



1. One-Stage Synthesis
   - 1 timing constraint
2. Two-Stage Synthesis
   - *Relaxed* constraint for selection
   - *Desired* constraint for balancing

Implementation & Flow Strategy

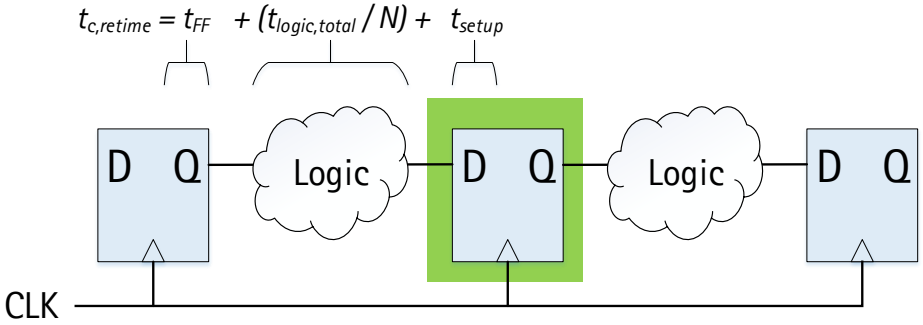


2. Pipeline
   - Described behaviorally by output registration
   - Tool performs retiming/register balancing

VHDL

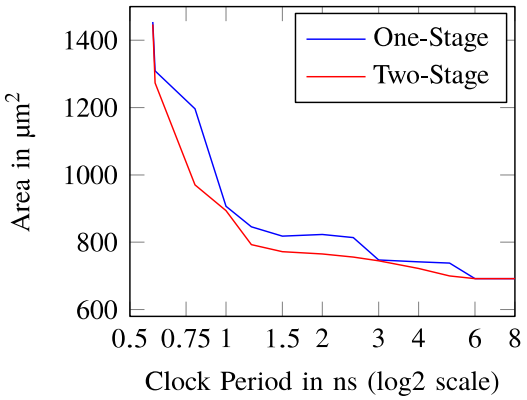
```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Synthesis



1. One-Stage Synthesis
   - 1 timing constraint
2. Two-Stage Synthesis
   - *Relaxed* constraint for selection
   - *Desired* constraint for balancing

Implementation & Flow Strategy

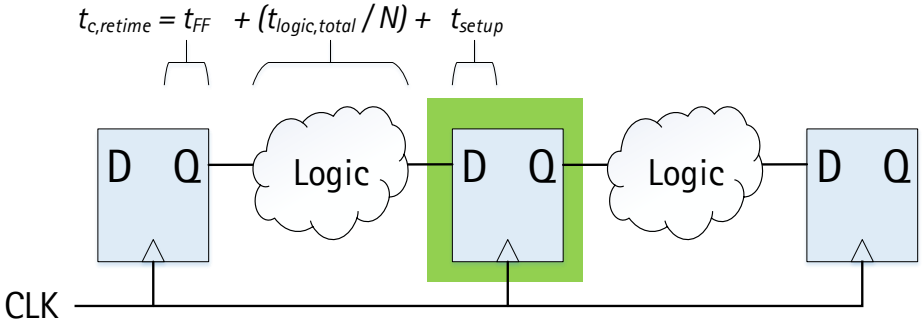


2. Pipeline
   - Described behaviorally by output registration
   - Tool performs retiming/register balancing

VHDL

```
C_tmp <= A + B;
process(clk)
begin
  if rising_edge(clk) then
    C <= C_tmp;
  end if;
end process;
```

Synthesis



1. One-Stage Synthesis
   - 1 timing constraint
2. Two-Stage Synthesis
   - *Relaxed* constraint for selection
   - *Desired* constraint for balancing

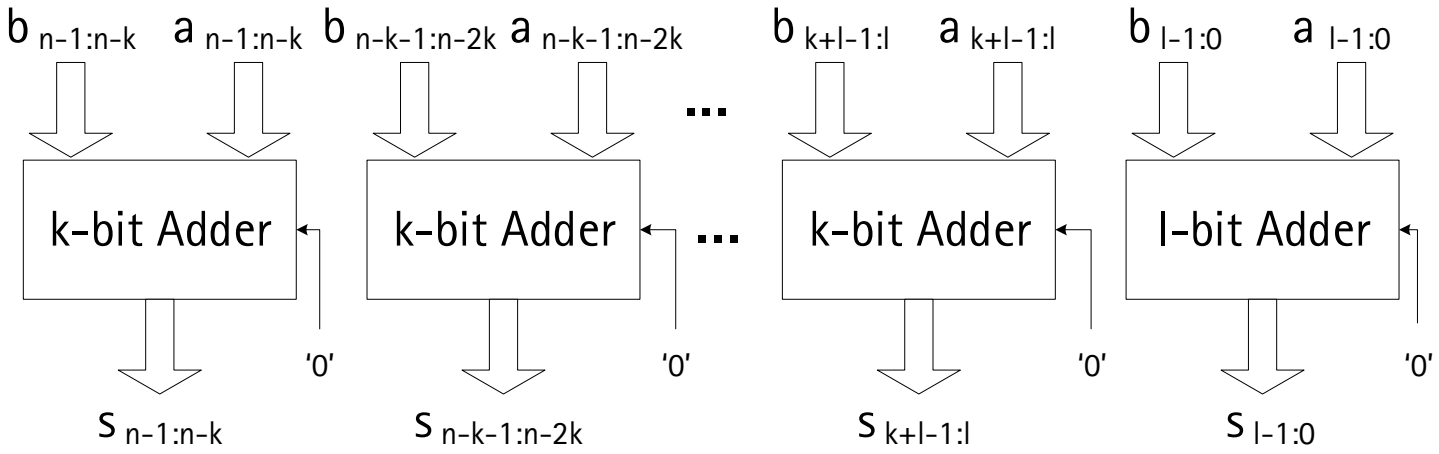
Implementation: Approximate Adders

- Precise adders in VHDL inferred using ⊕ operator
  - Precise reference architecture: *Parallel-Prefix architecture (32b)*

Implementation: Approximate Adders

- Precise adders in VHDL inferred using ⊕ operator
  - Precise reference architecture: *Parallel-Prefix architecture (32b)*

ESA

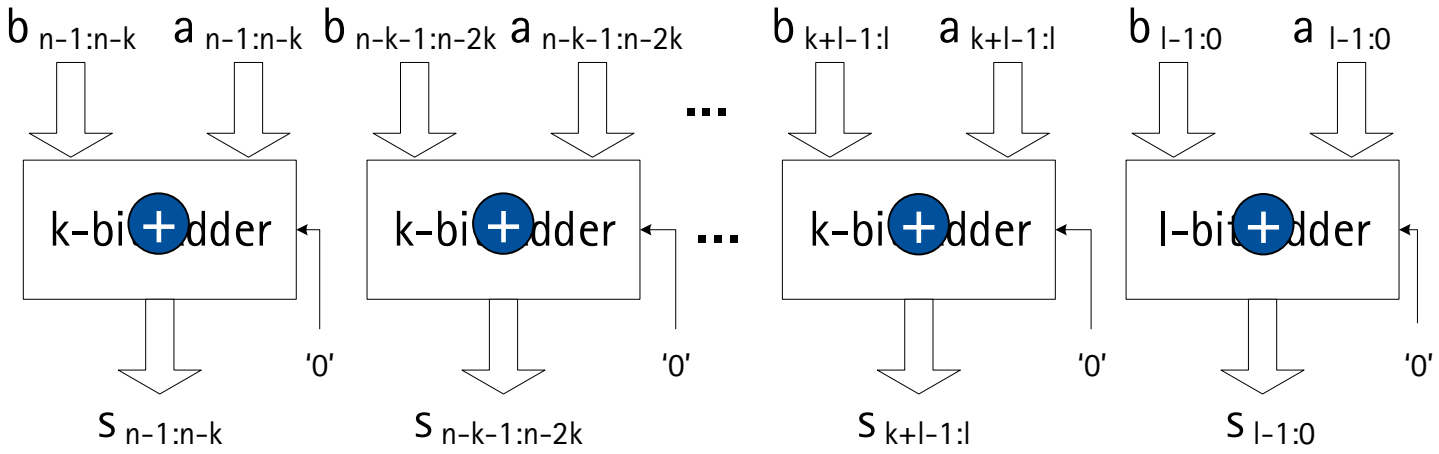


Mohapatra et al., *Design of voltage-scalable meta-functions for approximate computing*, 2011

Implementation: Approximate Adders

- Precise adders in VHDL inferred using + operator
  - Precise reference architecture: *Parallel-Prefix architecture (32b)*

ESA

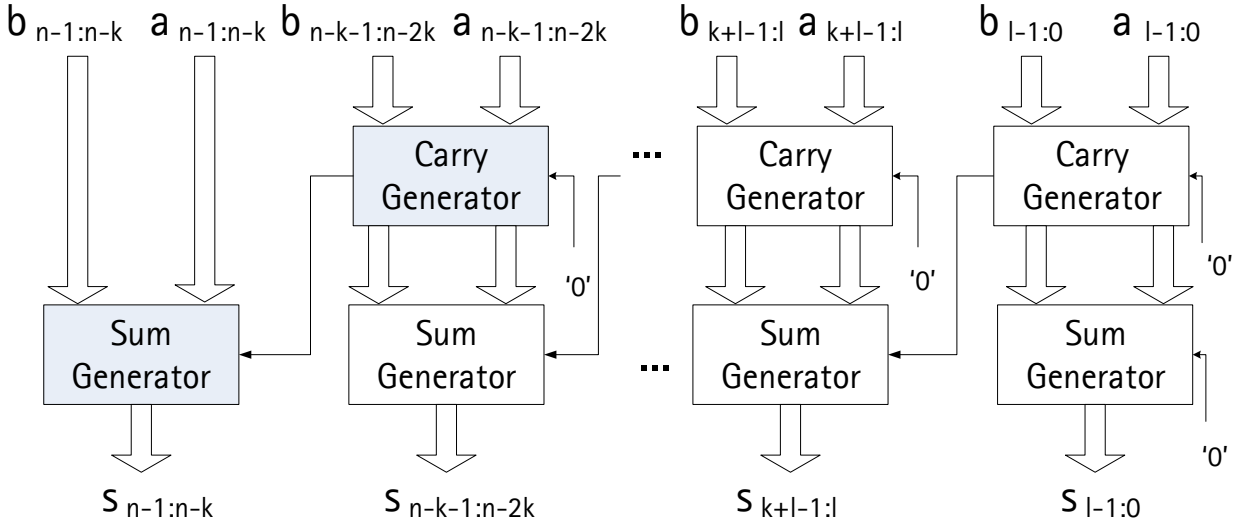


Mohapatra et al., *Design of voltage-scalable meta-functions for approximate computing*, 2011

Implementation: Approximate Adders

- Precise adders in VHDL inferred using ⊕ operator
  - Precise reference architecture: *Parallel-Prefix architecture (32b)*

ETA2

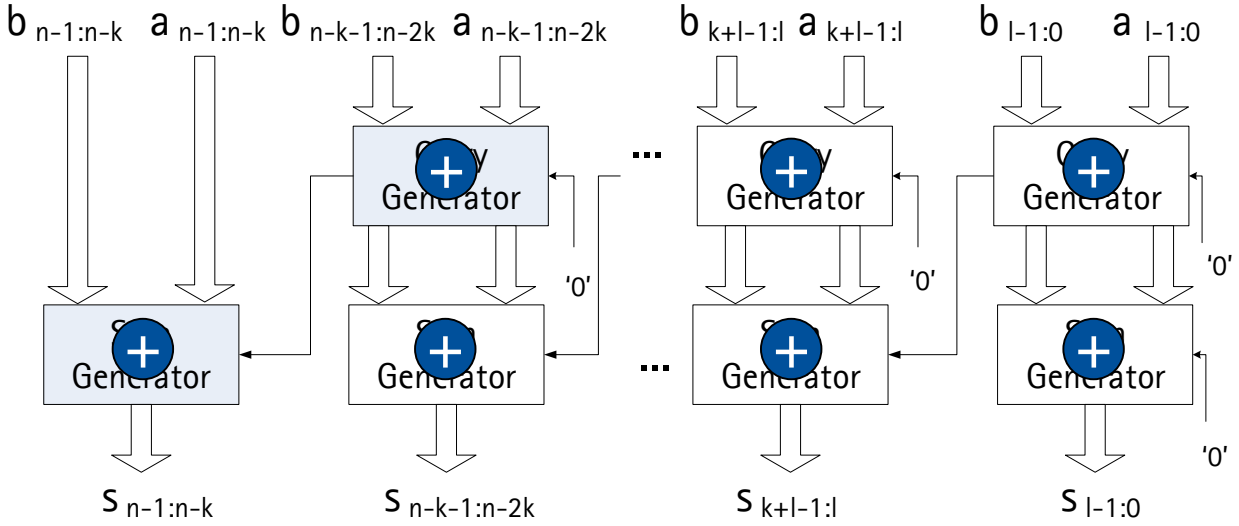


Zhu et al., *An enhanced low-power high-speed adder for error-tolerant application*, 2009

Implementation: Approximate Adders

- Precise adders in VHDL inferred using ⊕ operator
  - Precise reference architecture: *Parallel-Prefix architecture (32b)*

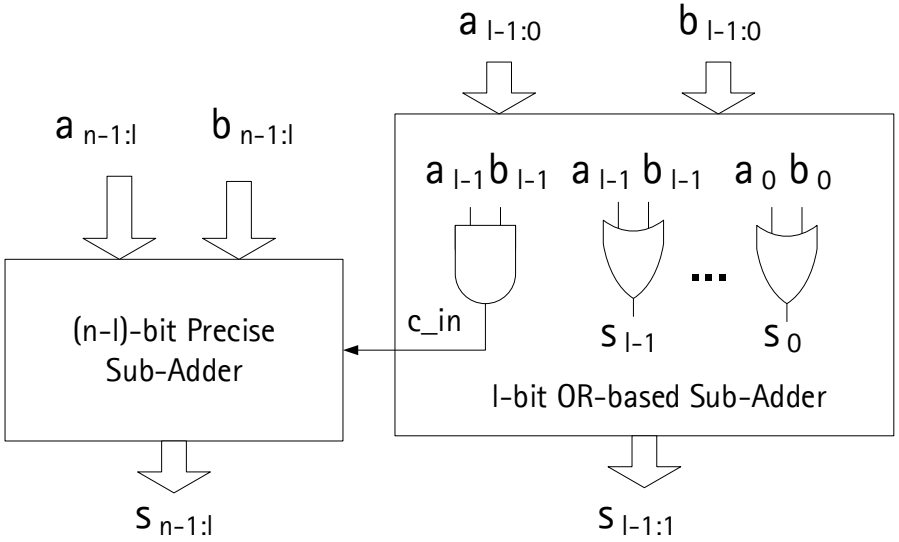
ETA2



Zhu et al., *An enhanced low-power high-speed adder for error-tolerant application*, 2009

Implementation: Approximate Adders

- Precise adders in VHDL inferred using ⊕ operator
  - Precise reference architecture: *Parallel-Prefix architecture (32b)*

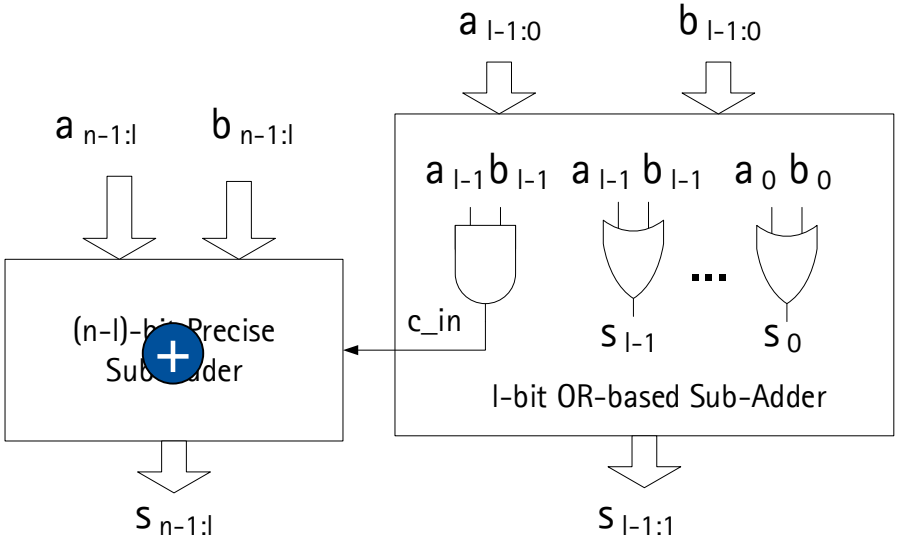


LOA

Mahdiani et al., *Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications*, 2010

Implementation: Approximate Adders

- Precise adders in VHDL inferred using ⊕ operator
  - Precise reference architecture: *Parallel-Prefix architecture (32b)*



LOA

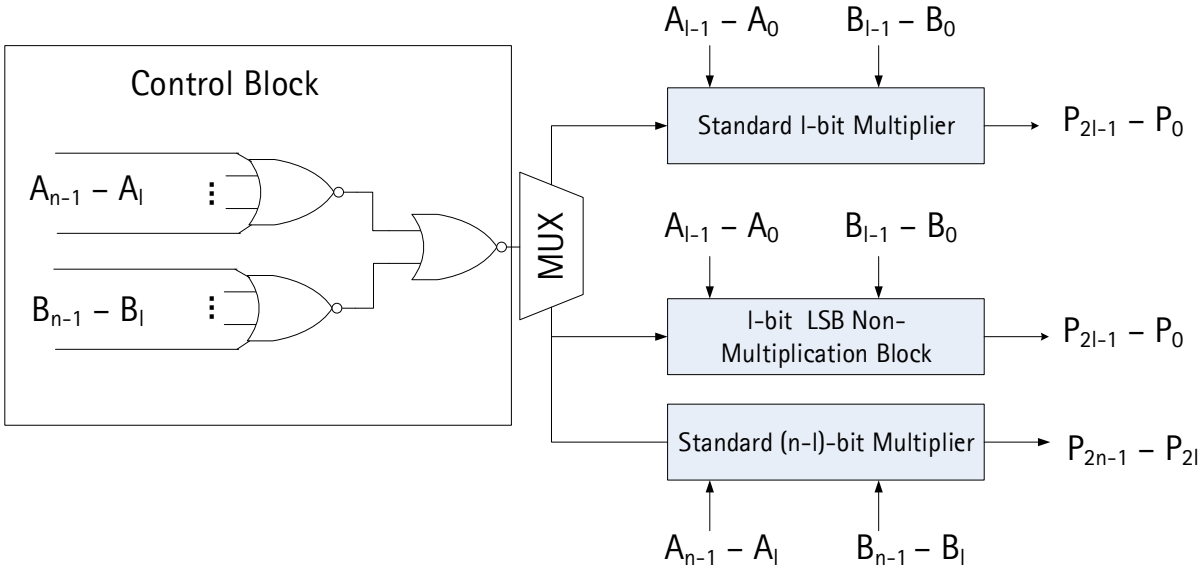
Mahdiani et al., *Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications*, 2010

Implementation: Approximate Multipliers

- Precise multipliers in VHDL inferred using * operator
  - Precise reference architecture: *Radix-4 Booth (32b)*

Implementation: Approximate Multipliers

- Precise multipliers in VHDL inferred using * operator
  - Precise reference architecture: *Radix-4 Booth (32b)*

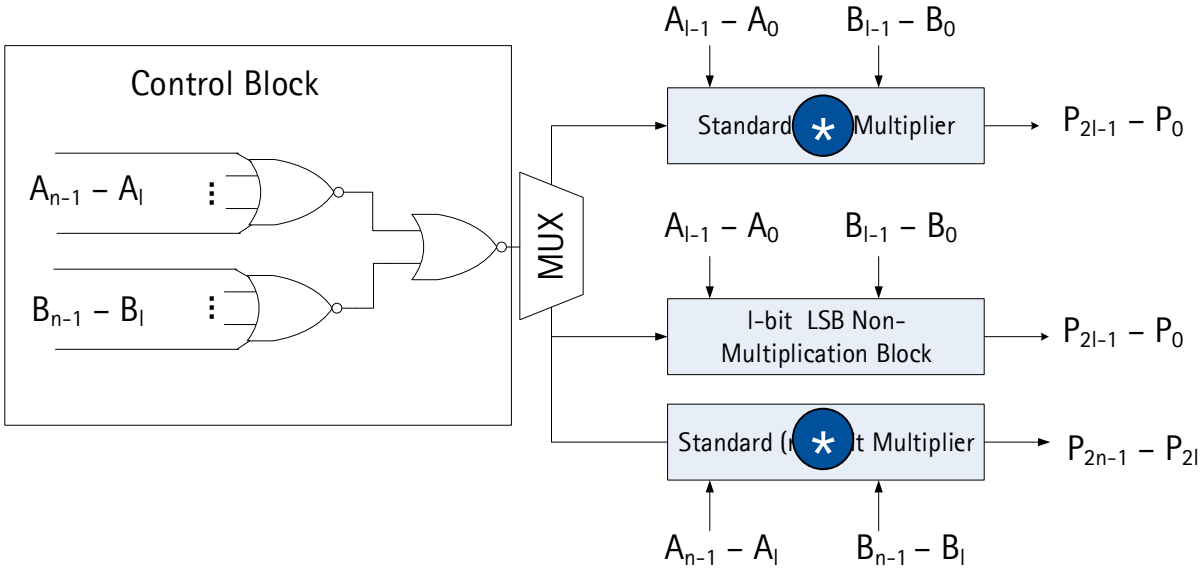


ETM

Kyaw et al., *Low-power high-speed multiplier for error-tolerant application*, 2010

Implementation: Approximate Multipliers

- Precise multipliers in VHDL inferred using * operator
  - Precise reference architecture: *Radix-4 Booth (32b)*

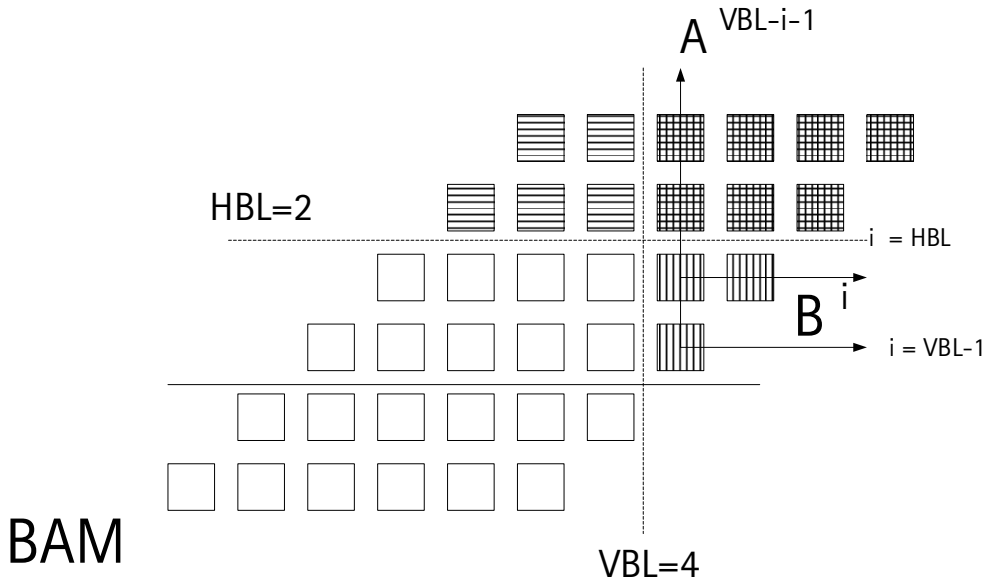


ETM

Kyaw et al., *Low-power high-speed multiplier for error-tolerant application*, 2010

Implementation: Approximate Multipliers

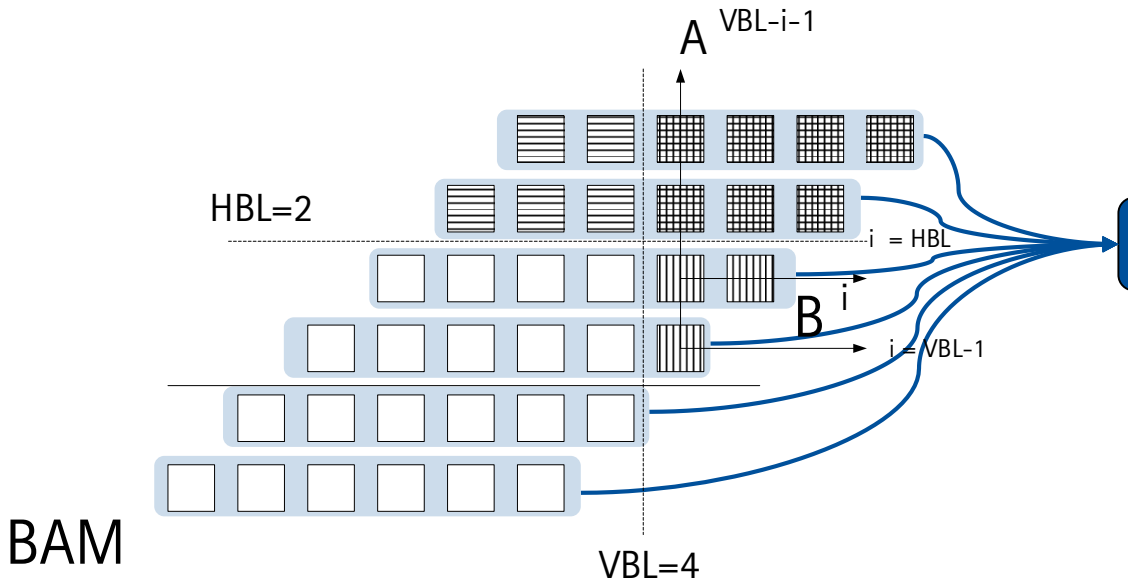
- Precise multipliers in VHDL inferred using * operator
  - Precise reference architecture: *Radix-4 Booth (32b)*



Mahdiani et al., *Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications*, 2010

Implementation: Approximate Multipliers

- Precise multipliers in VHDL inferred using * operator
  - Precise reference architecture: *Radix-4 Booth (32b)*



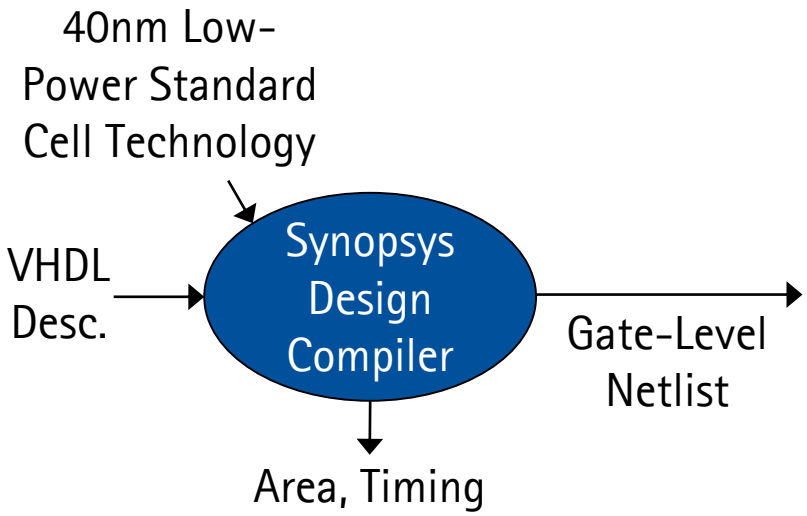
DWF_sum

Synopsys Design Compiler:
DesignWare library function for implementing an optimized carry-save adder tree

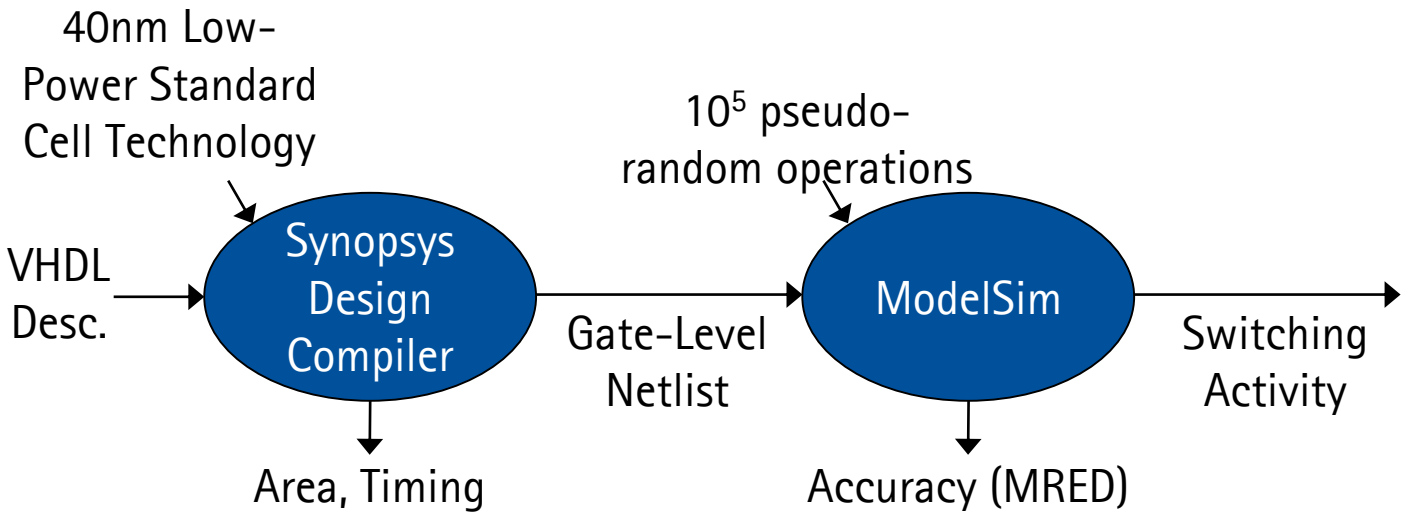
Mahdiani et al., *Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications*, 2010

ATE-Accuracy Profiling

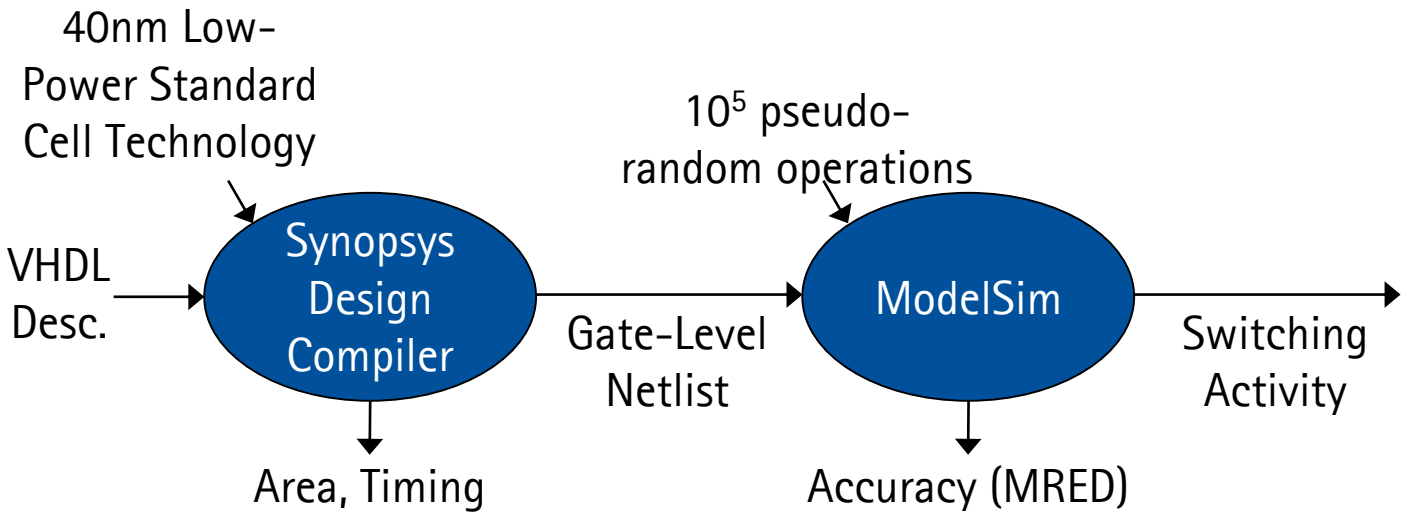
ATE-Accuracy Profiling



ATE-Accuracy Profiling



ATE-Accuracy Profiling



Relative Error Distance

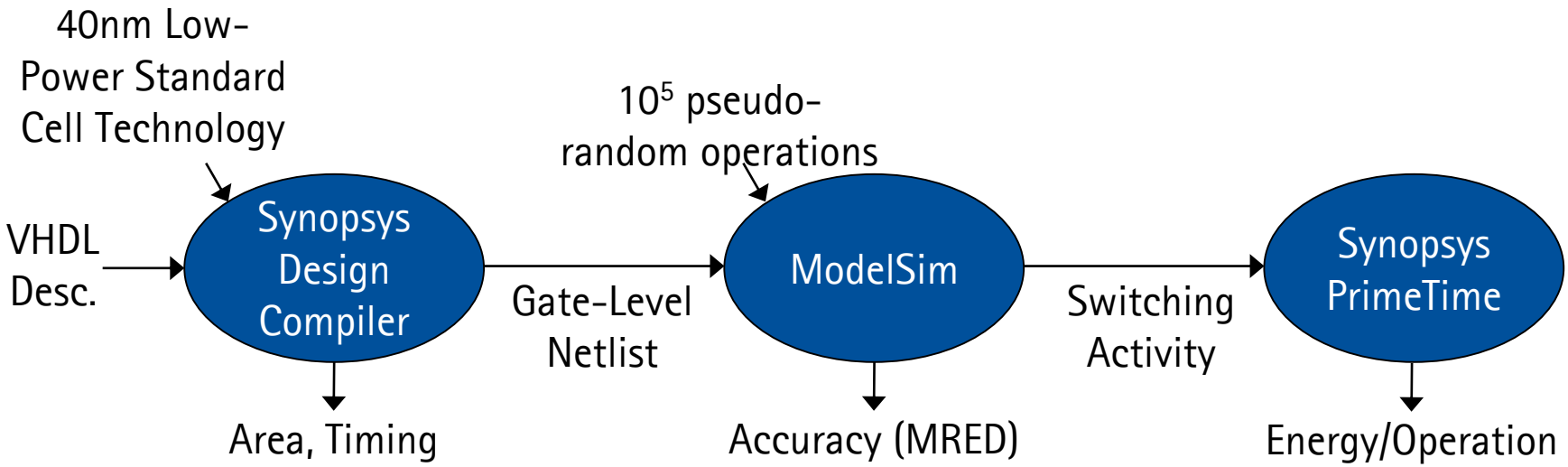
$$\mathrm{RED} = \frac{|M' - M|}{M}$$

M	Precise Result
M'	Approximate Result
n	Bitwidth
N	Number of pseudo-random operations

Mean Relative Error Distance

$$\mathrm{MRED} = E[\mathrm{RED}] = \frac{1}{2^{-2n}} \sum_{i=0}^{2^n-1} \sum_{j=0}^{2^n-1} \frac{|M'_{ij} - M_{ij}|}{M_{ij}} \approx \frac{1}{N} \sum_{i=0}^{N-1} \frac{|M'_i - M_i|}{M_i}$$

ATE-Accuracy Profiling



Relative Error Distance

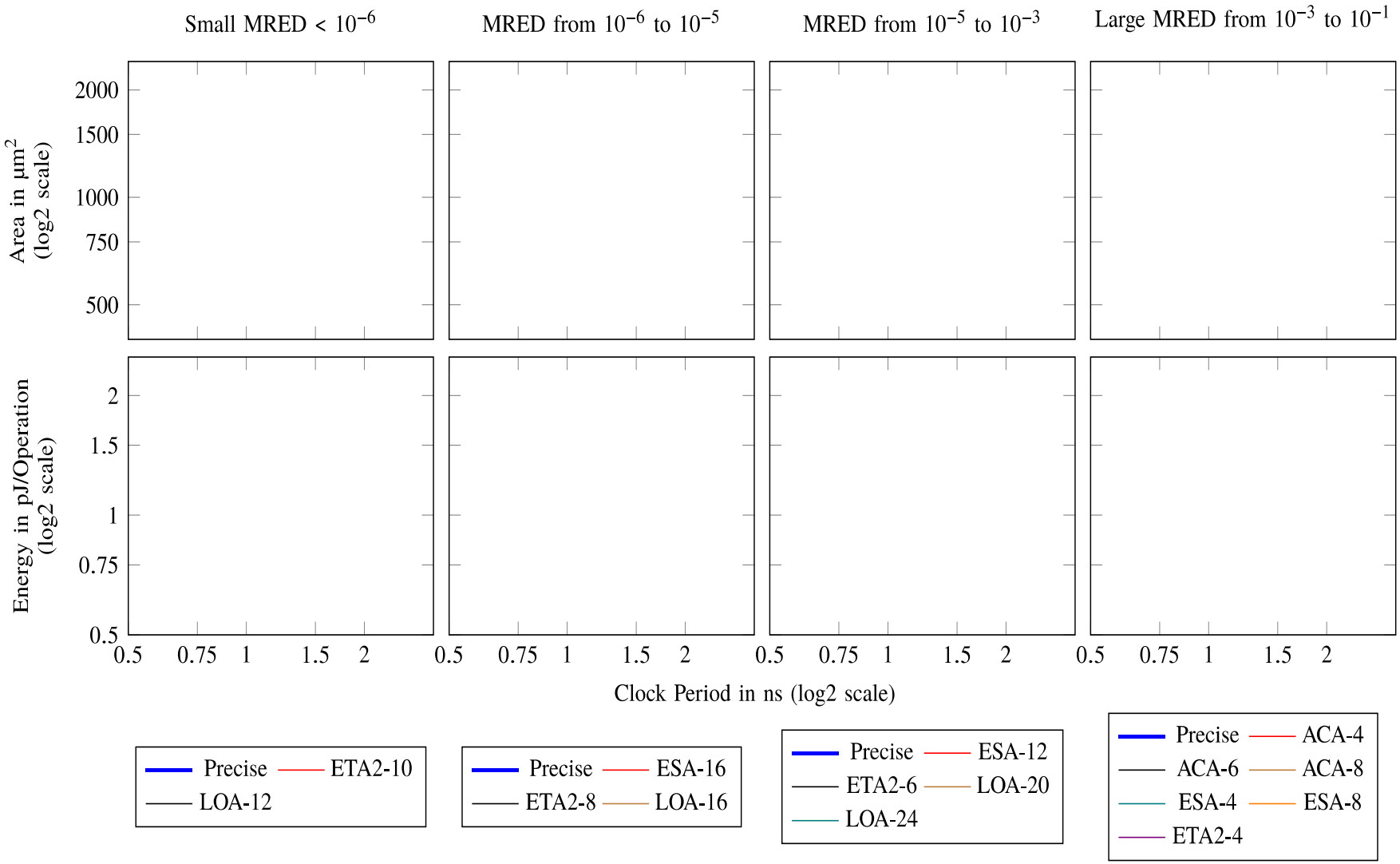
$$\mathrm{RED} = \frac{|M' - M|}{M}$$

M	Precise Result
M'	Approximate Result
n	Bitwidth
N	Number of pseudo-random operations

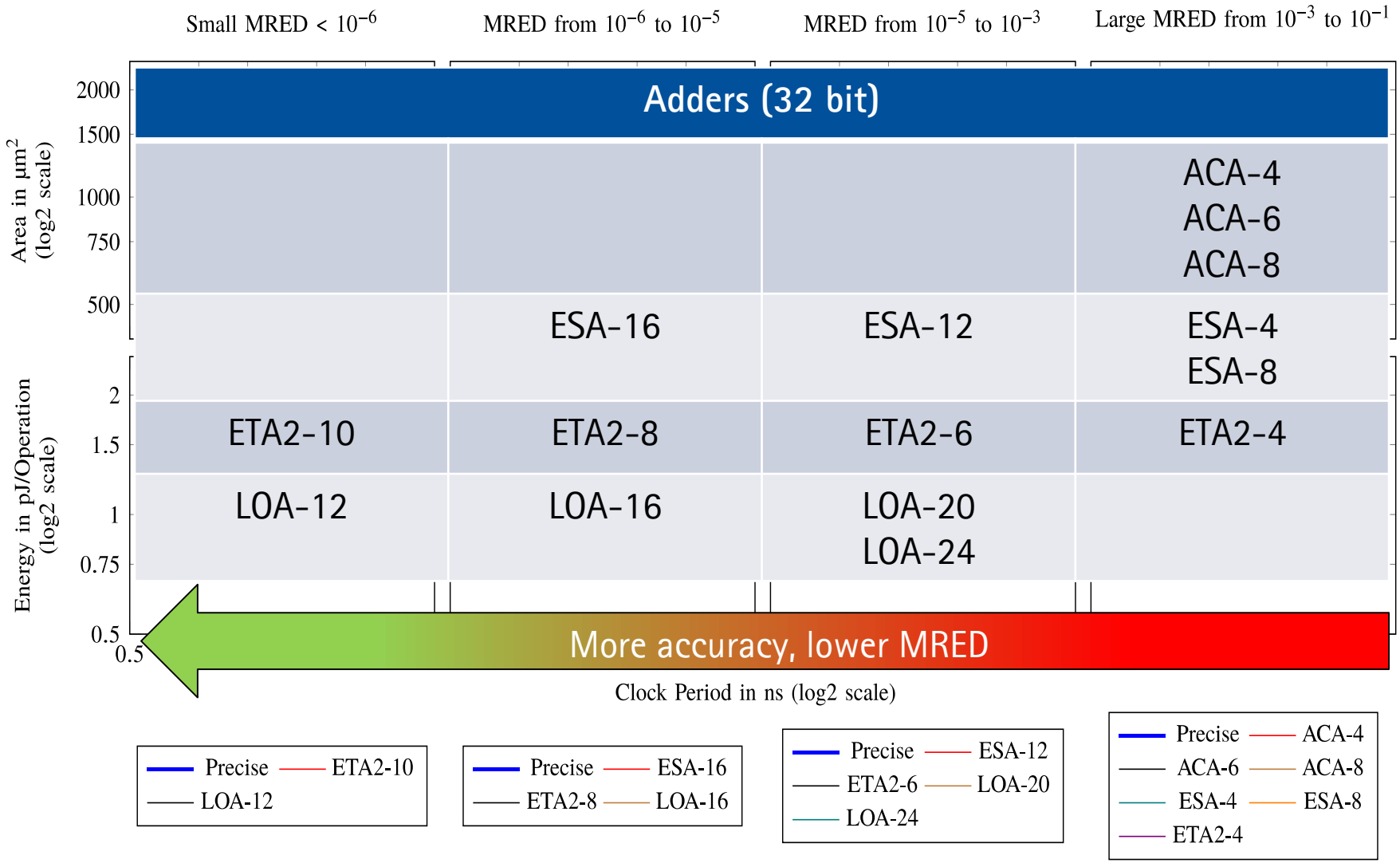
Mean Relative Error Distance

$$\mathrm{MRED} = E[\mathrm{RED}] = \frac{1}{2^{-2n}} \sum_{i=0}^{2^n-1} \sum_{j=0}^{2^n-1} \frac{|M'_{ij} - M_{ij}|}{M_{ij}} \approx \frac{1}{N} \sum_{i=0}^{N-1} \frac{|M'_i - M_i|}{M_i}$$

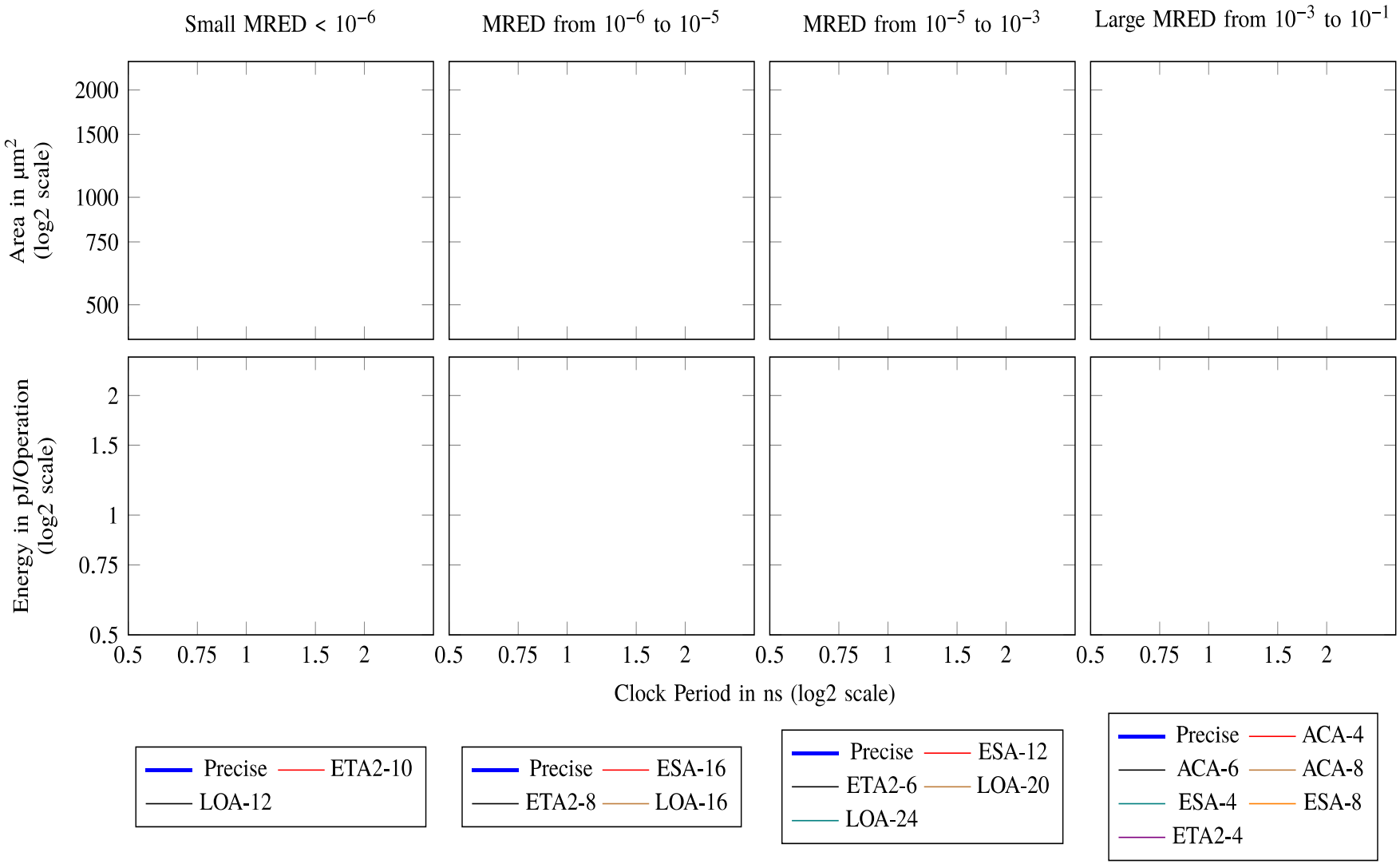
ATE-Accuracy Profiling: Approximate Adders



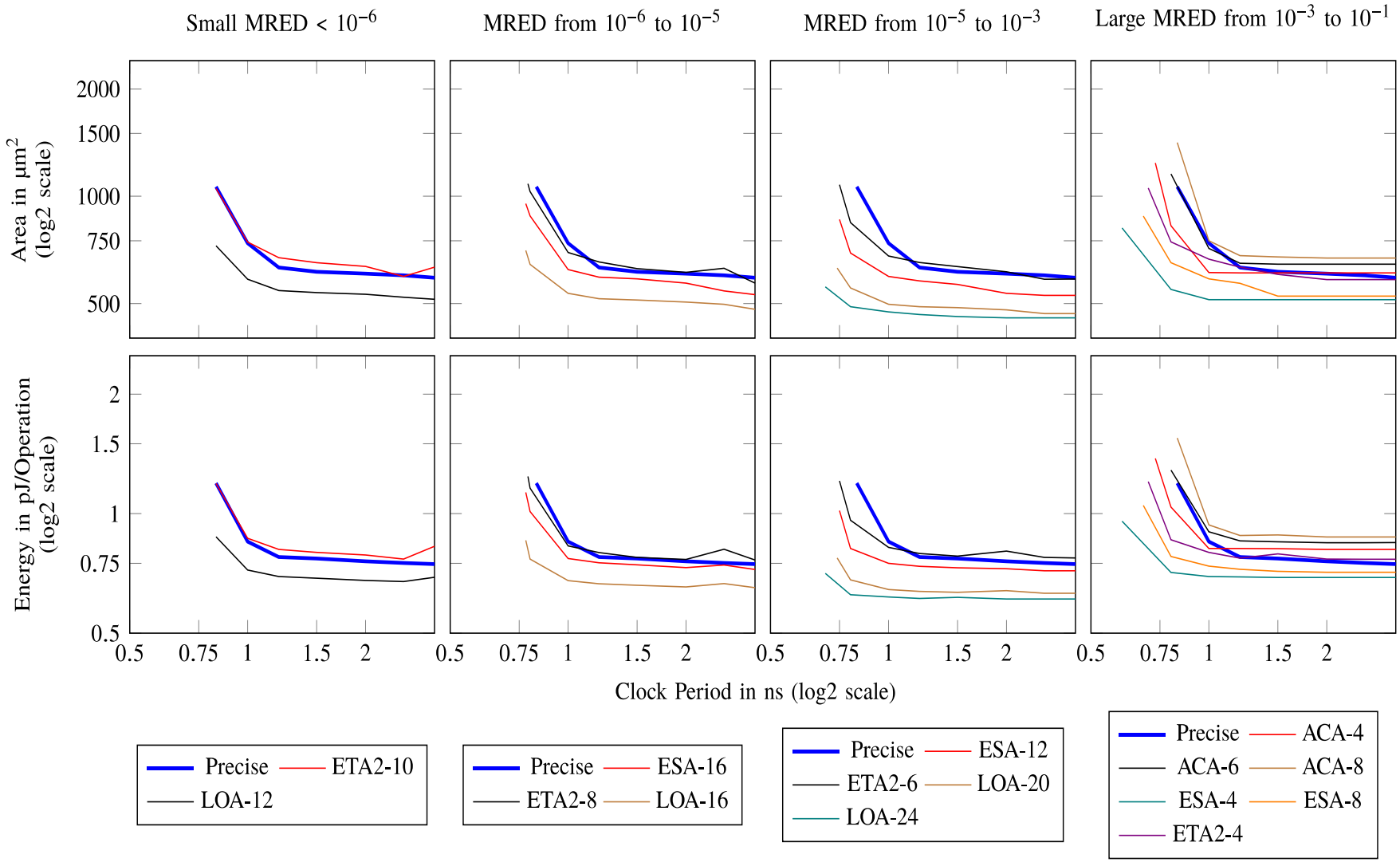
ATE-Accuracy Profiling: Approximate Adders



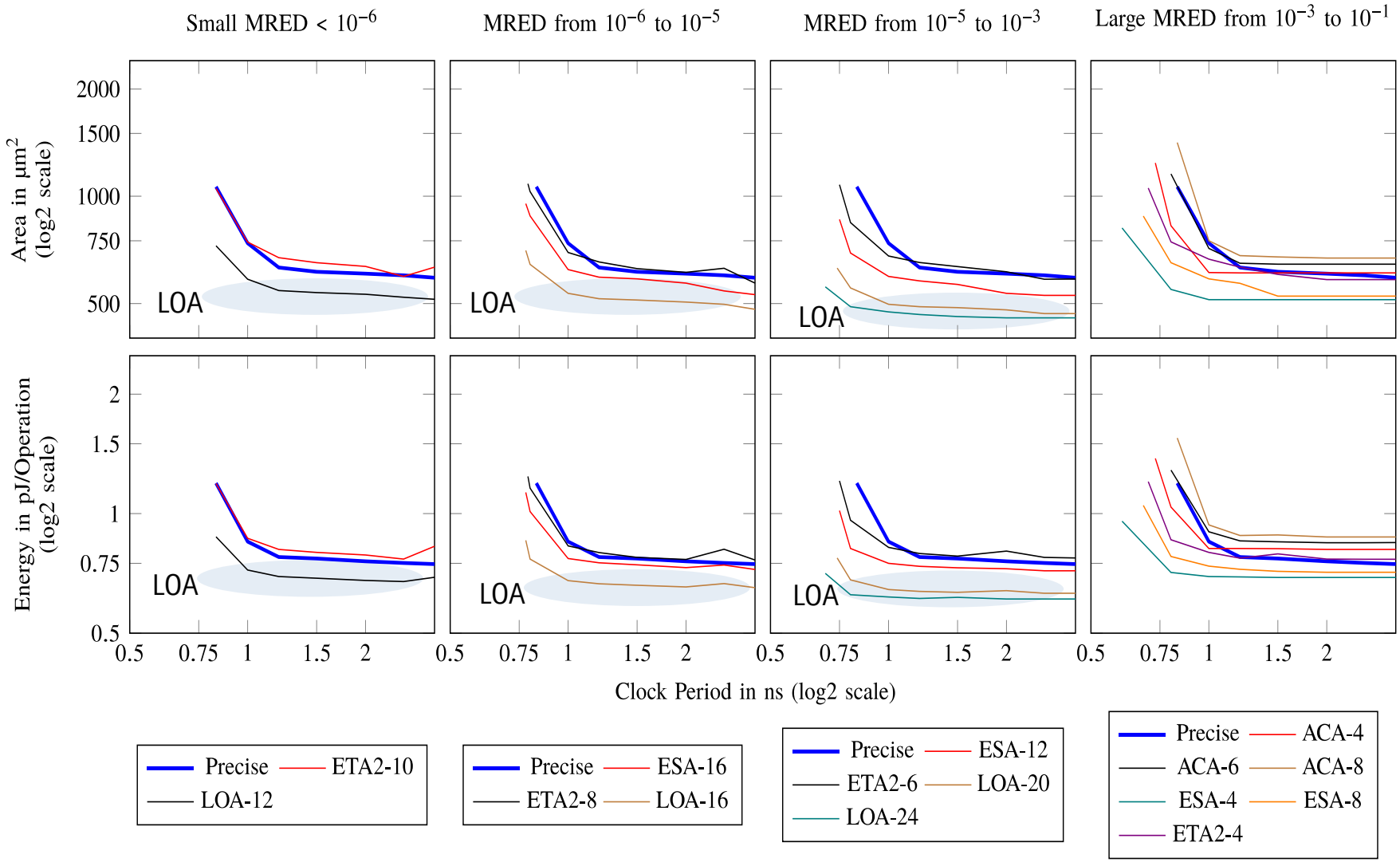
ATE-Accuracy Profiling: Approximate Adders



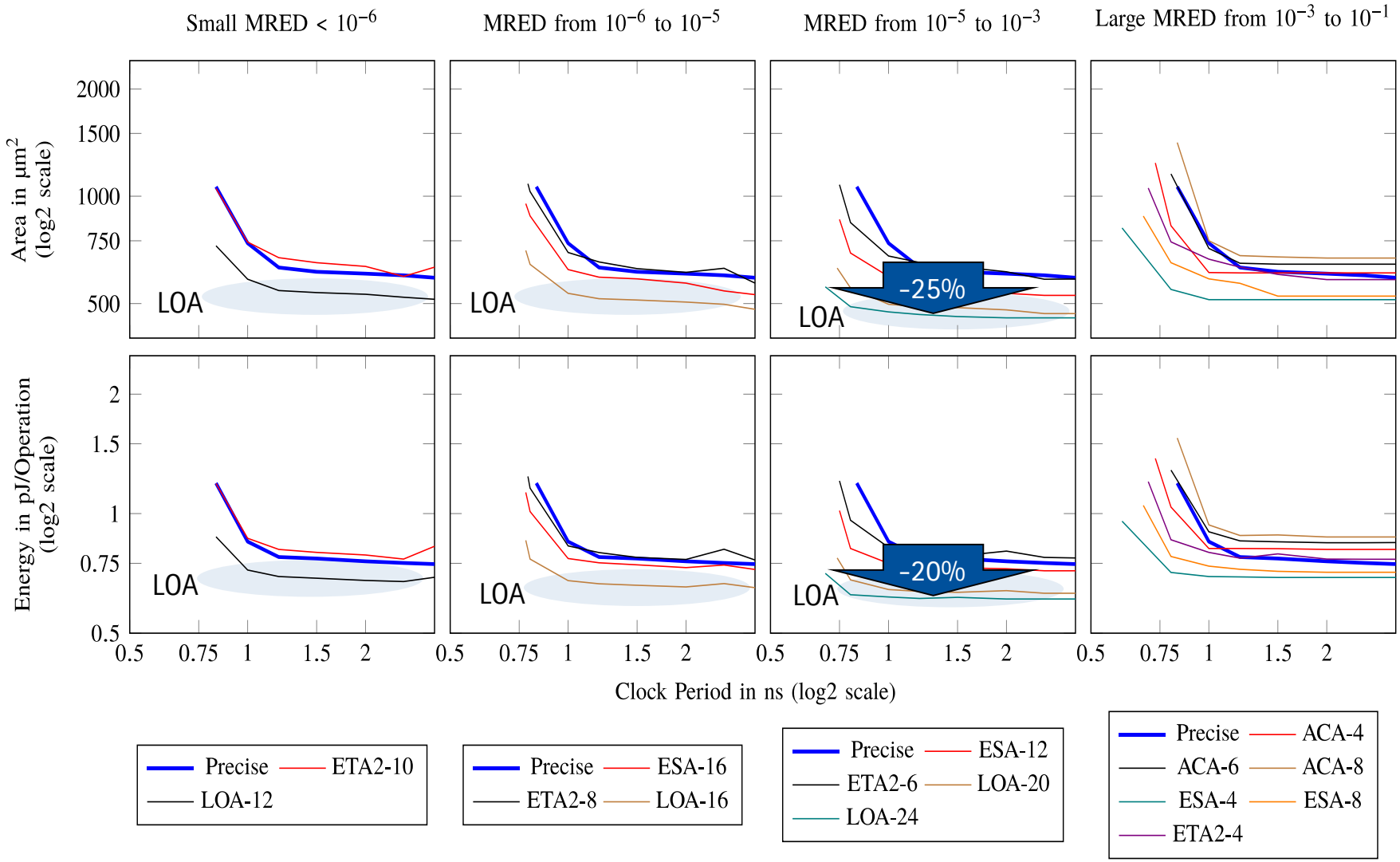
ATE-Accuracy Profiling: Approximate Adders



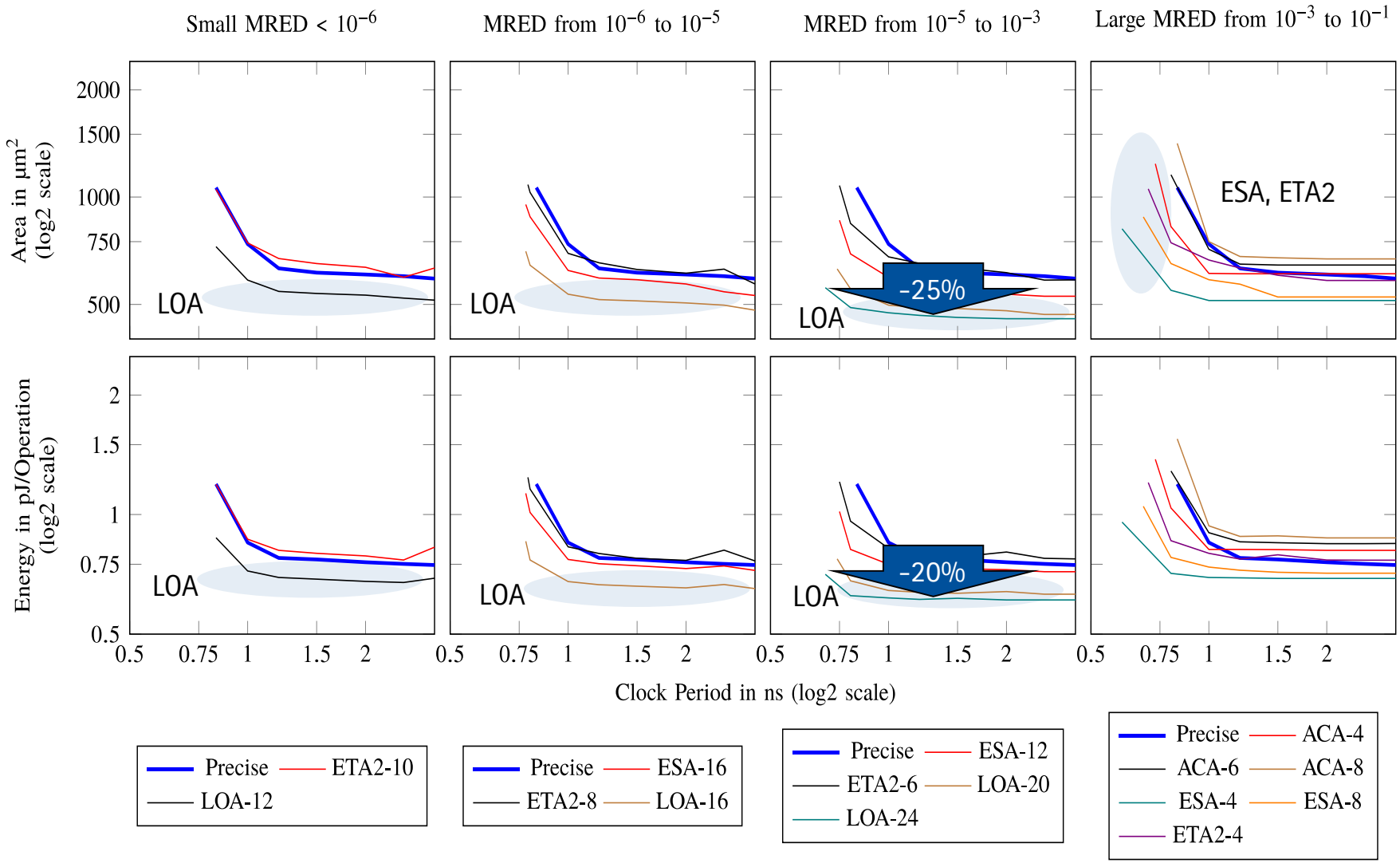
ATE-Accuracy Profiling: Approximate Adders



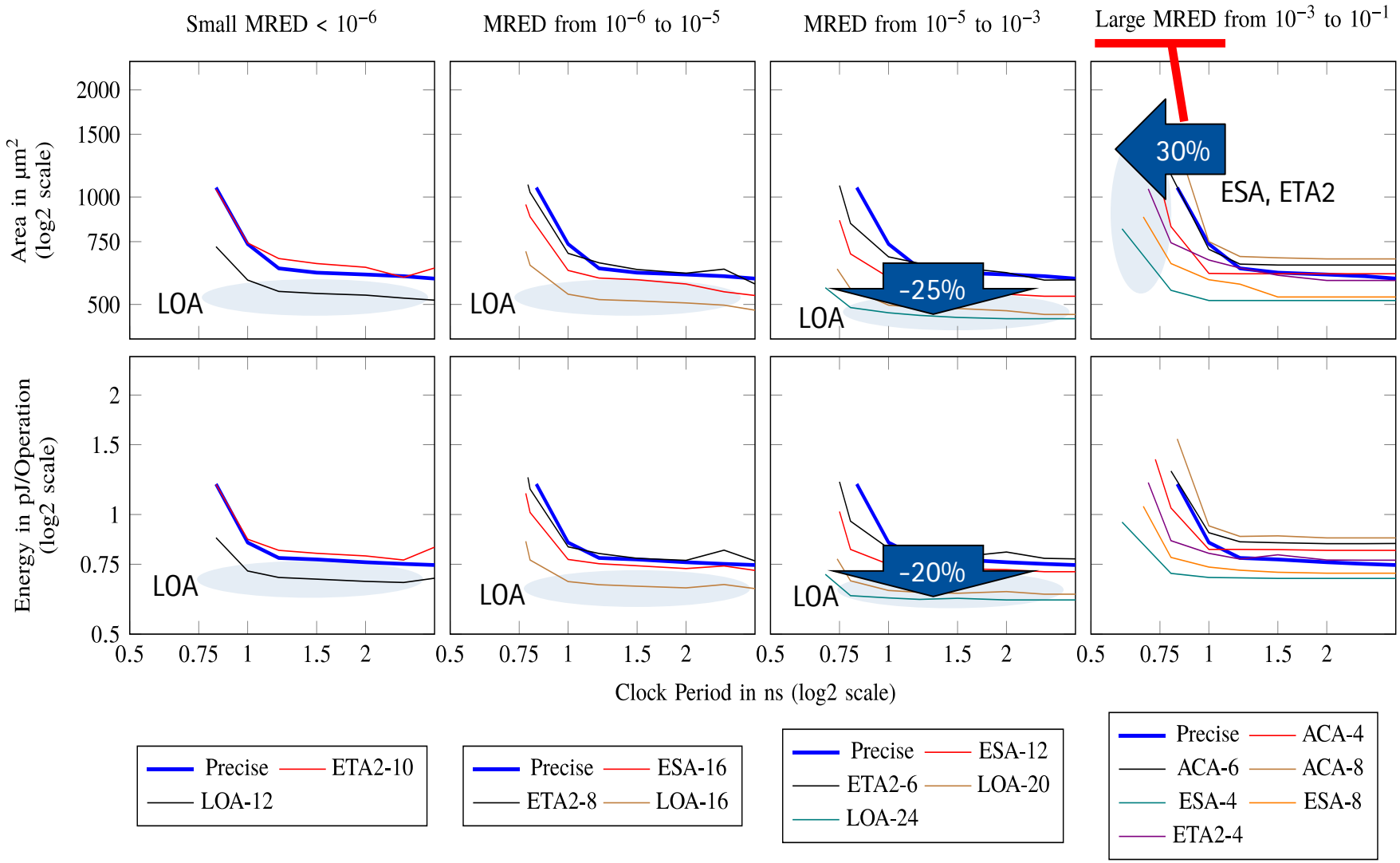
ATE-Accuracy Profiling: Approximate Adders



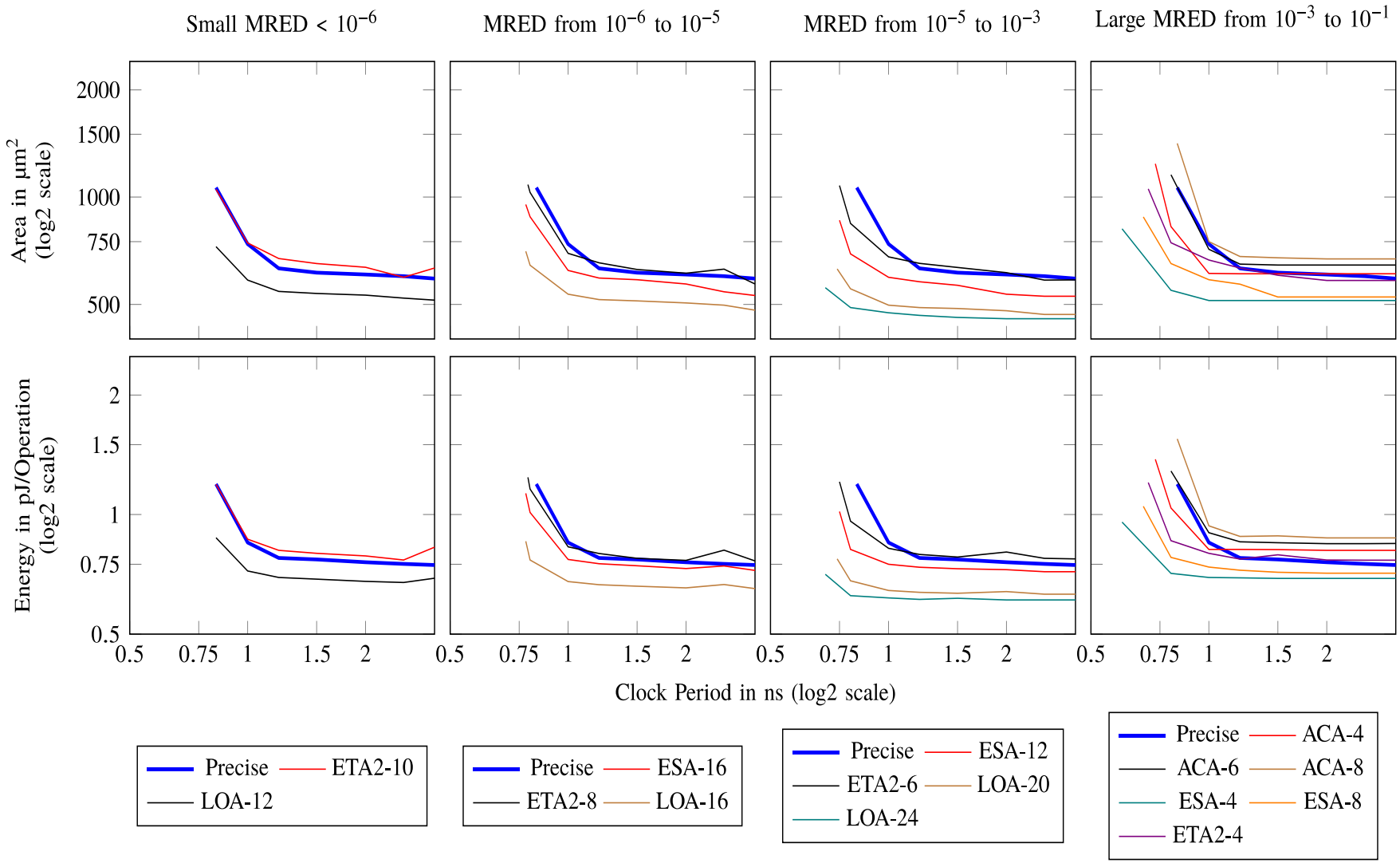
ATE-Accuracy Profiling: Approximate Adders



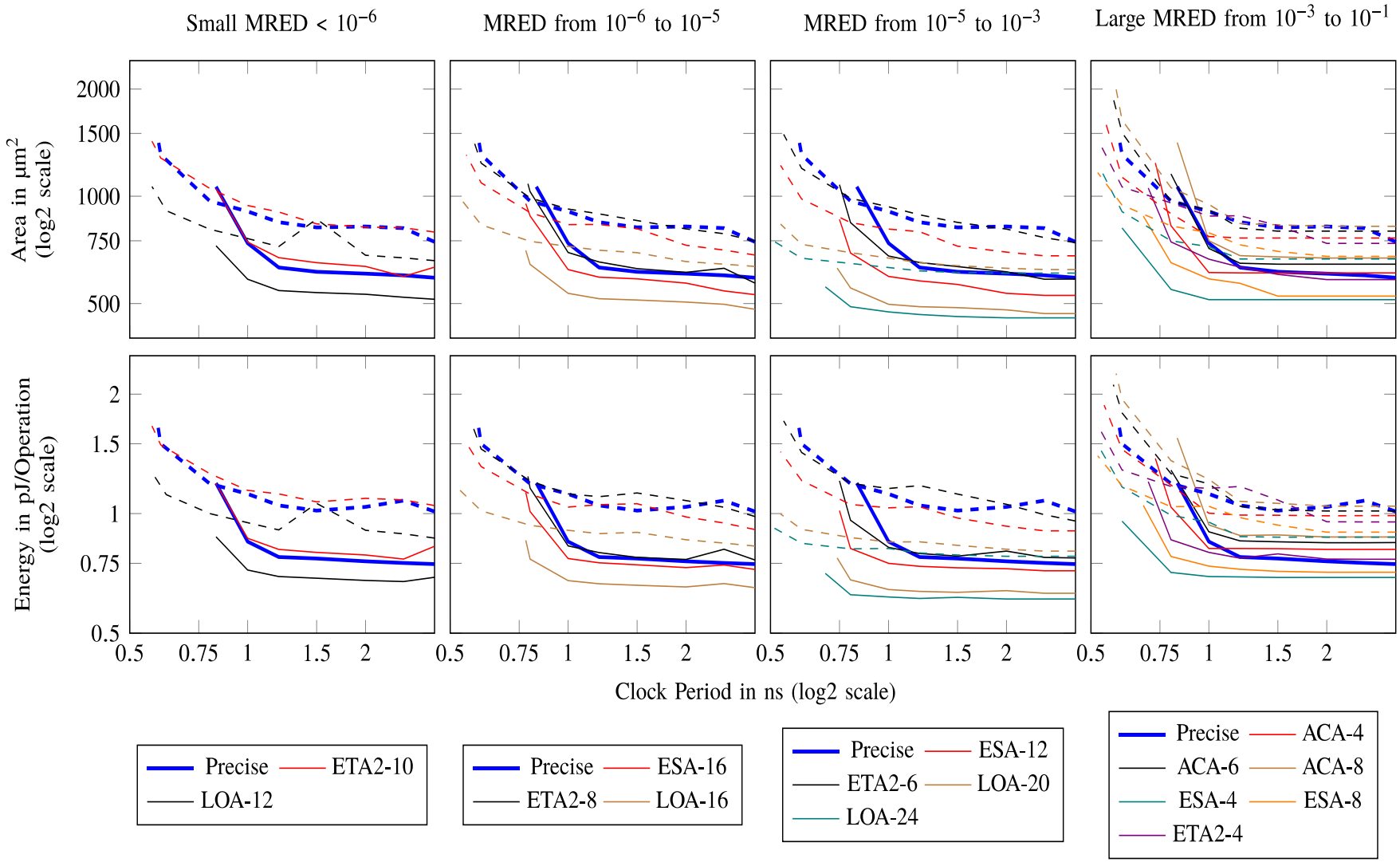
ATE-Accuracy Profiling: Approximate Adders



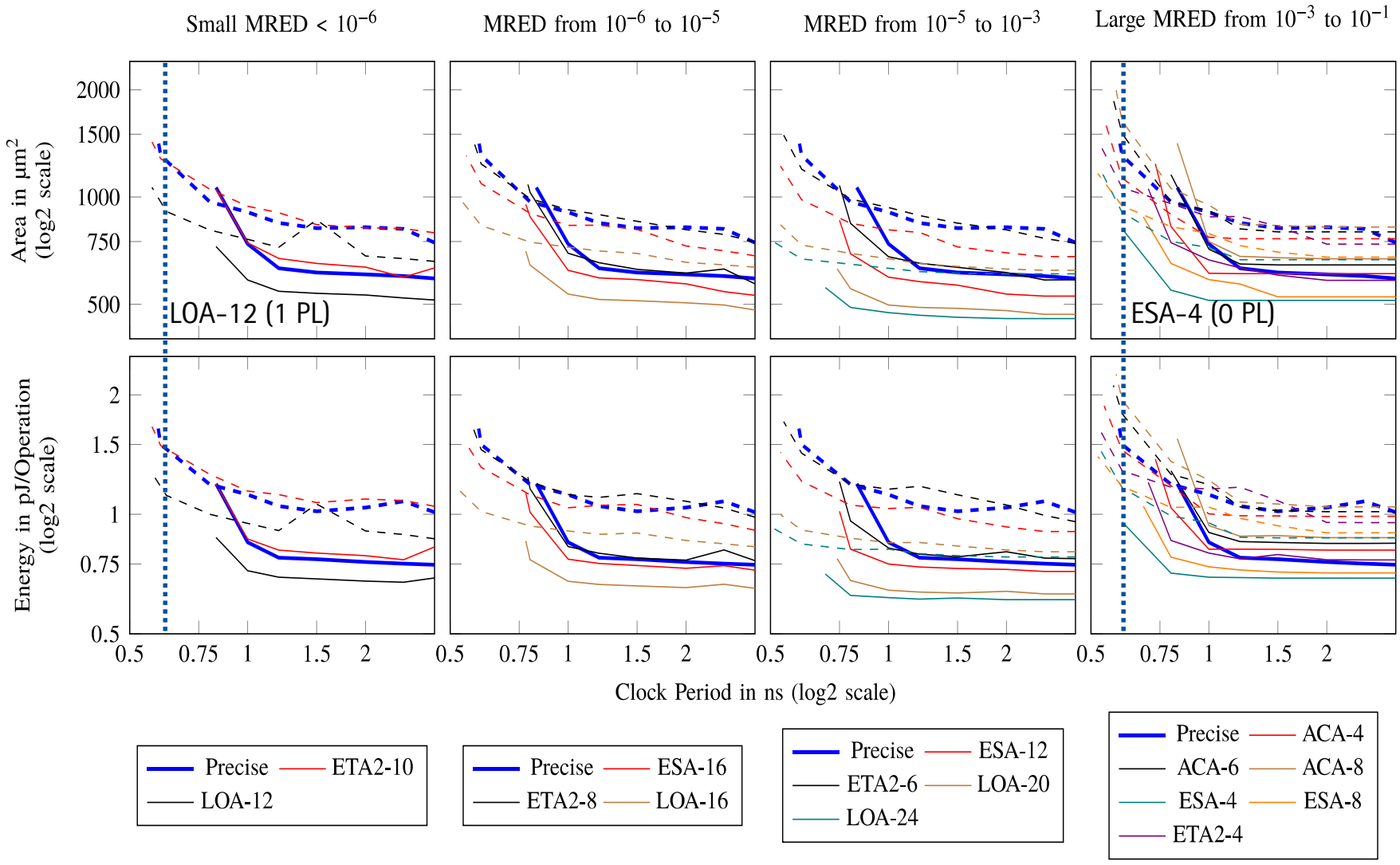
ATE-Accuracy Profiling: Approximate Adders



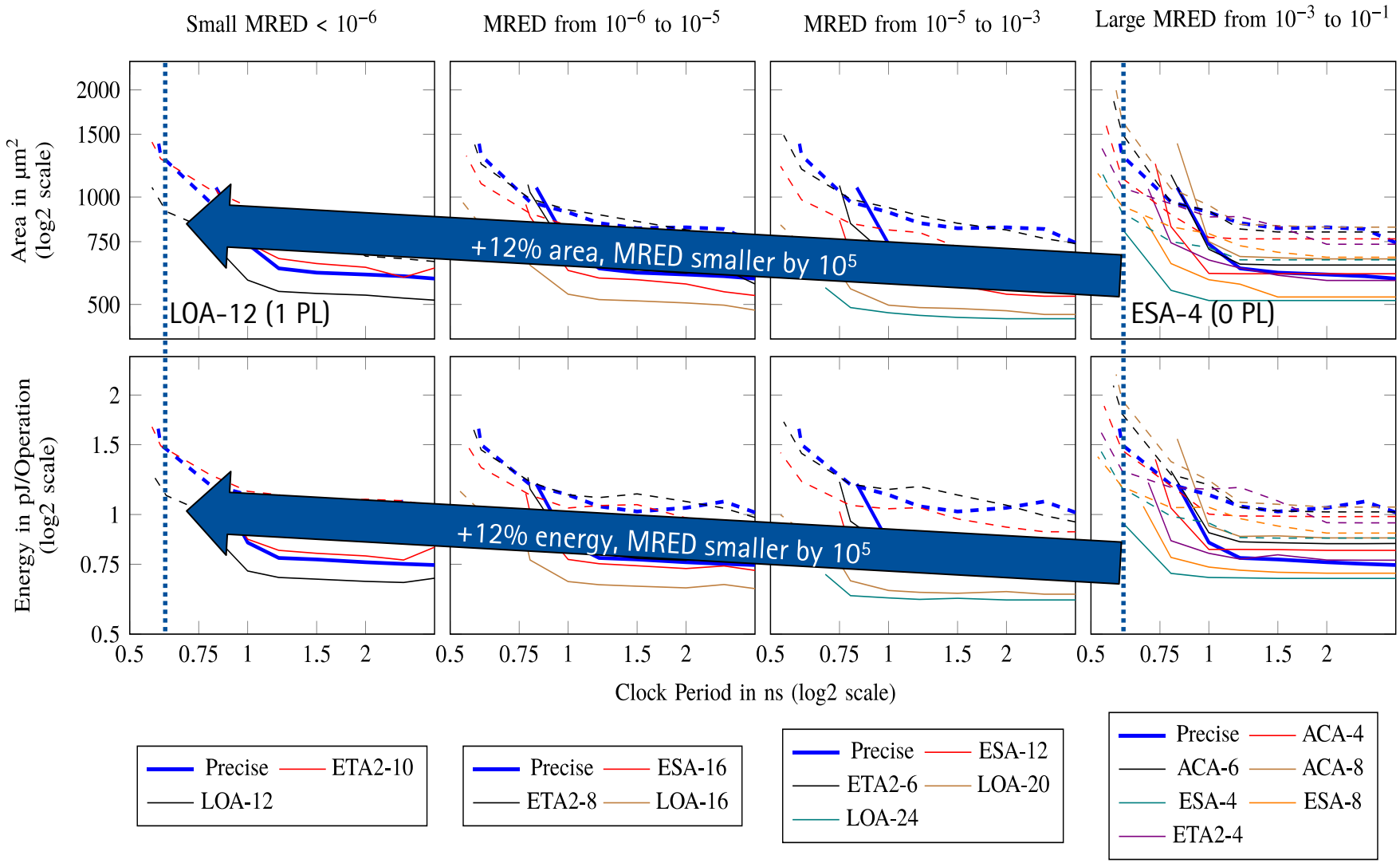
ATE-Accuracy Profiling: Approximate Adders



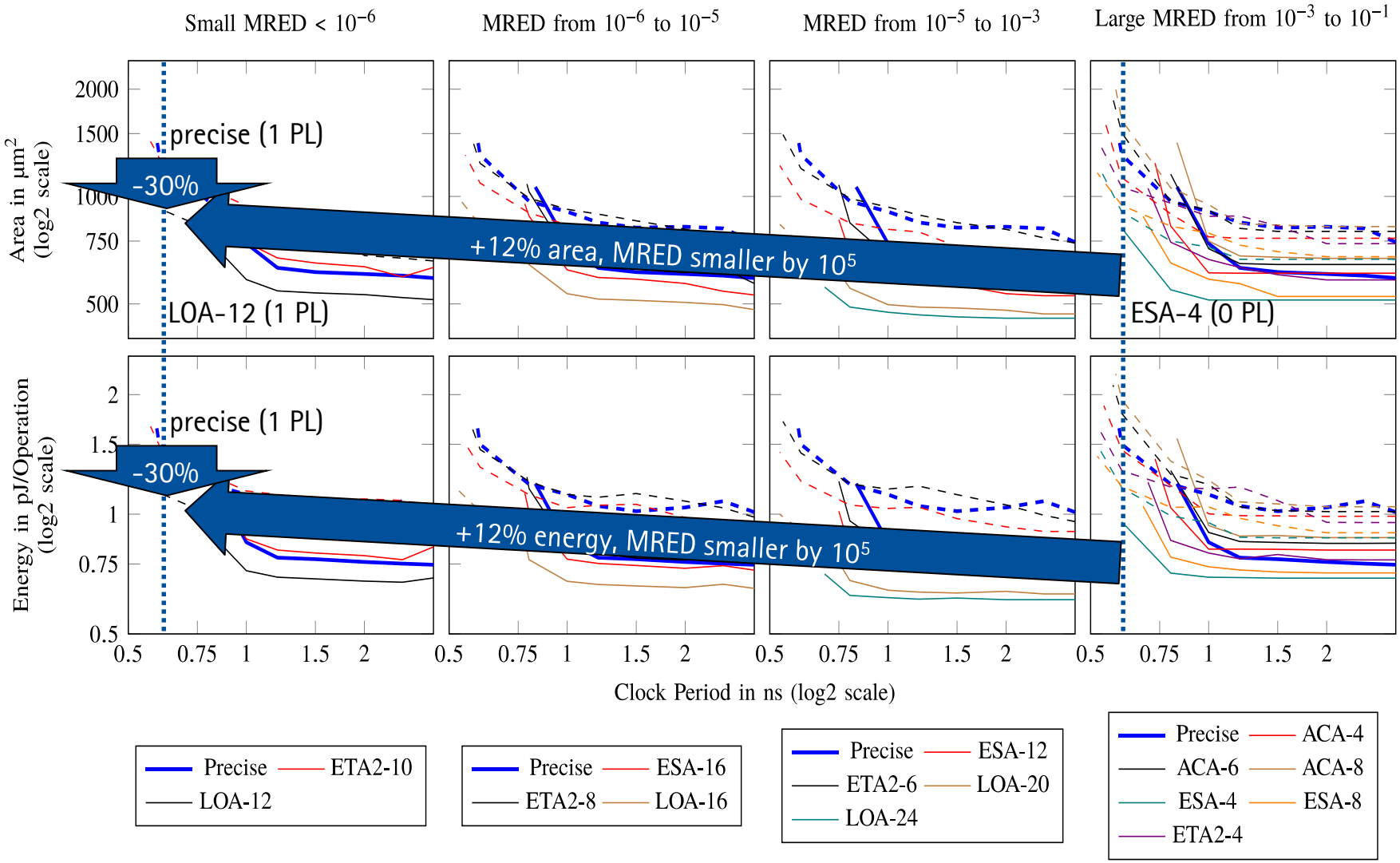
ATE-Accuracy Profiling: Approximate Adders



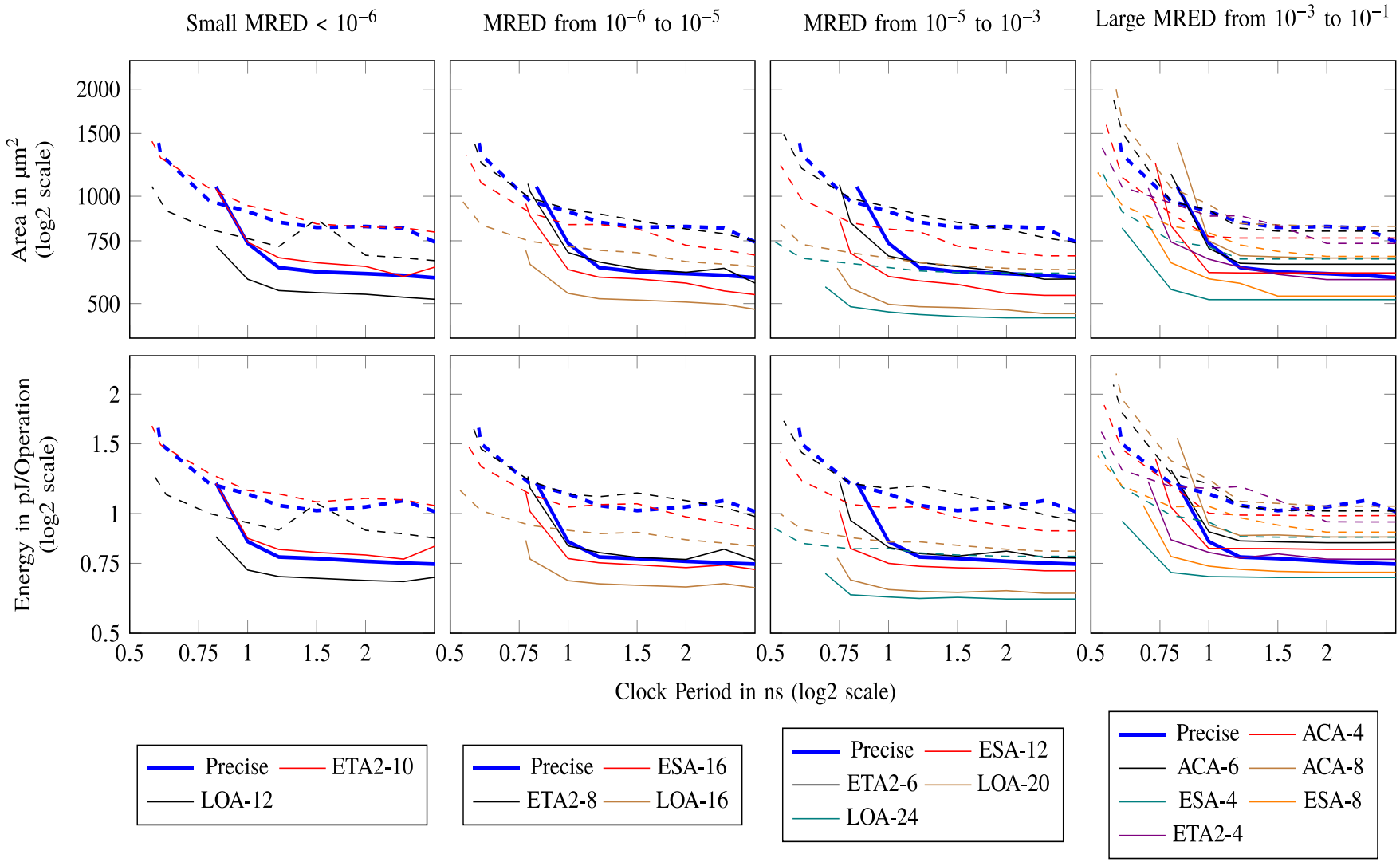
ATE-Accuracy Profiling: Approximate Adders



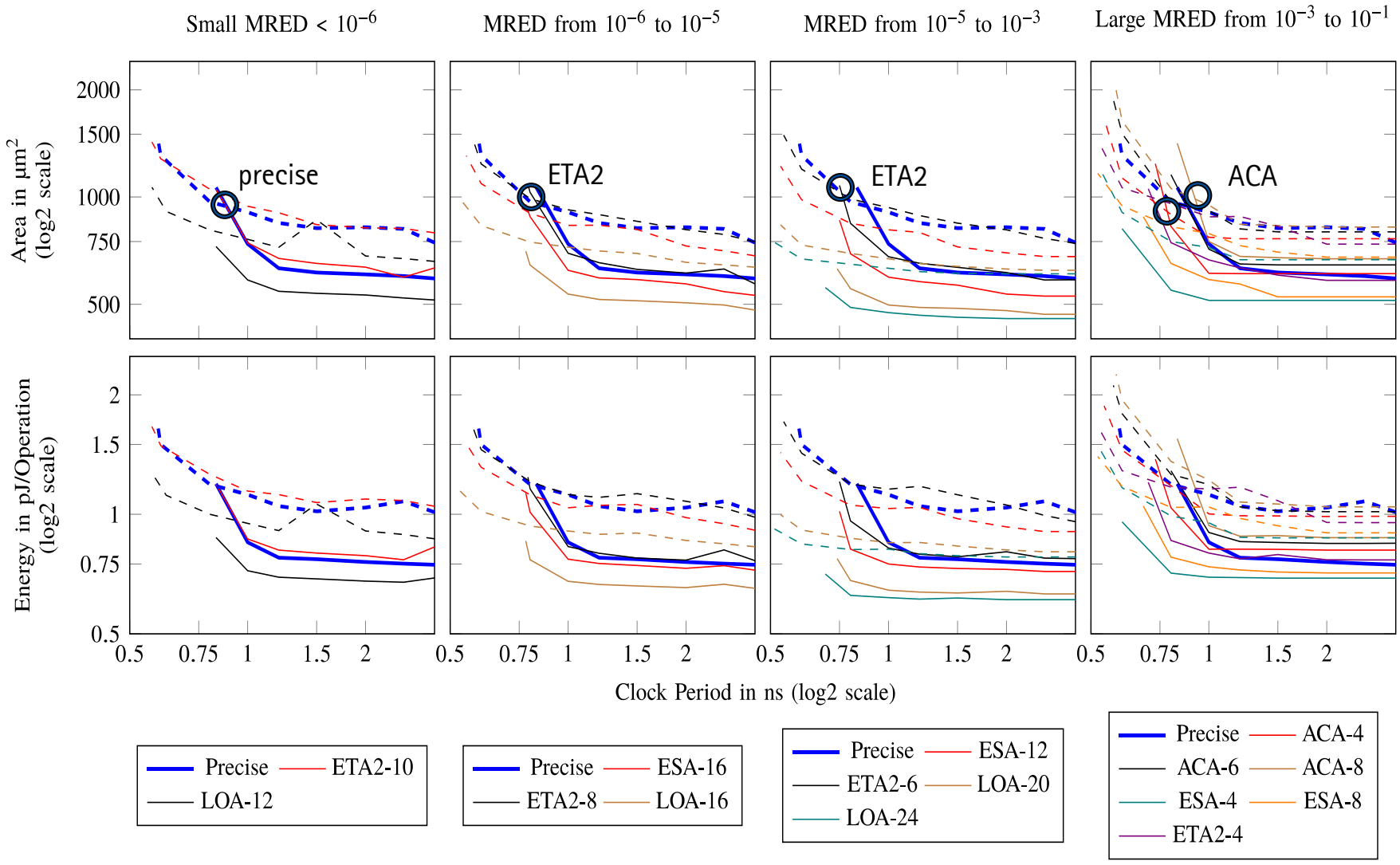
ATE-Accuracy Profiling: Approximate Adders



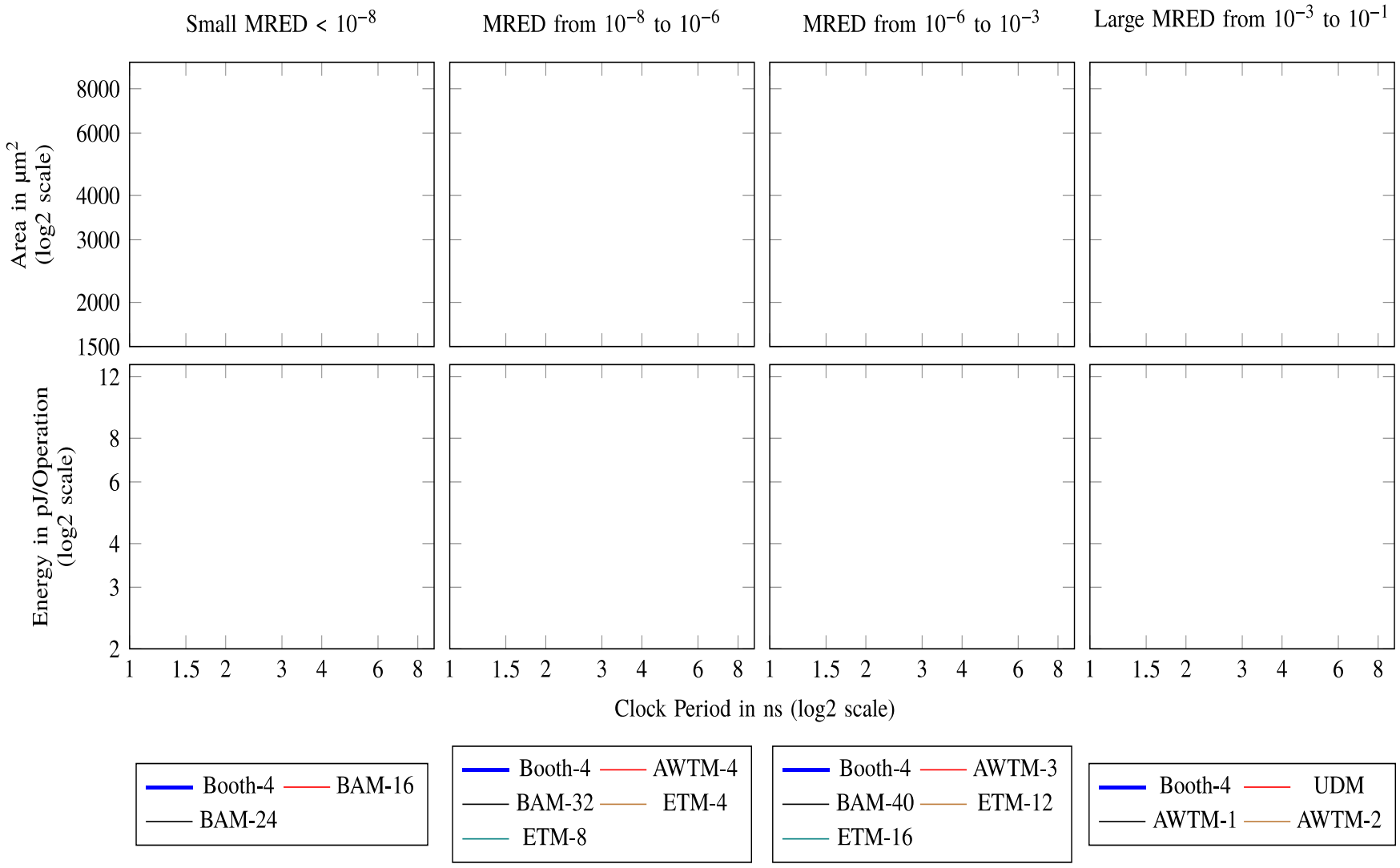
ATE-Accuracy Profiling: Approximate Adders



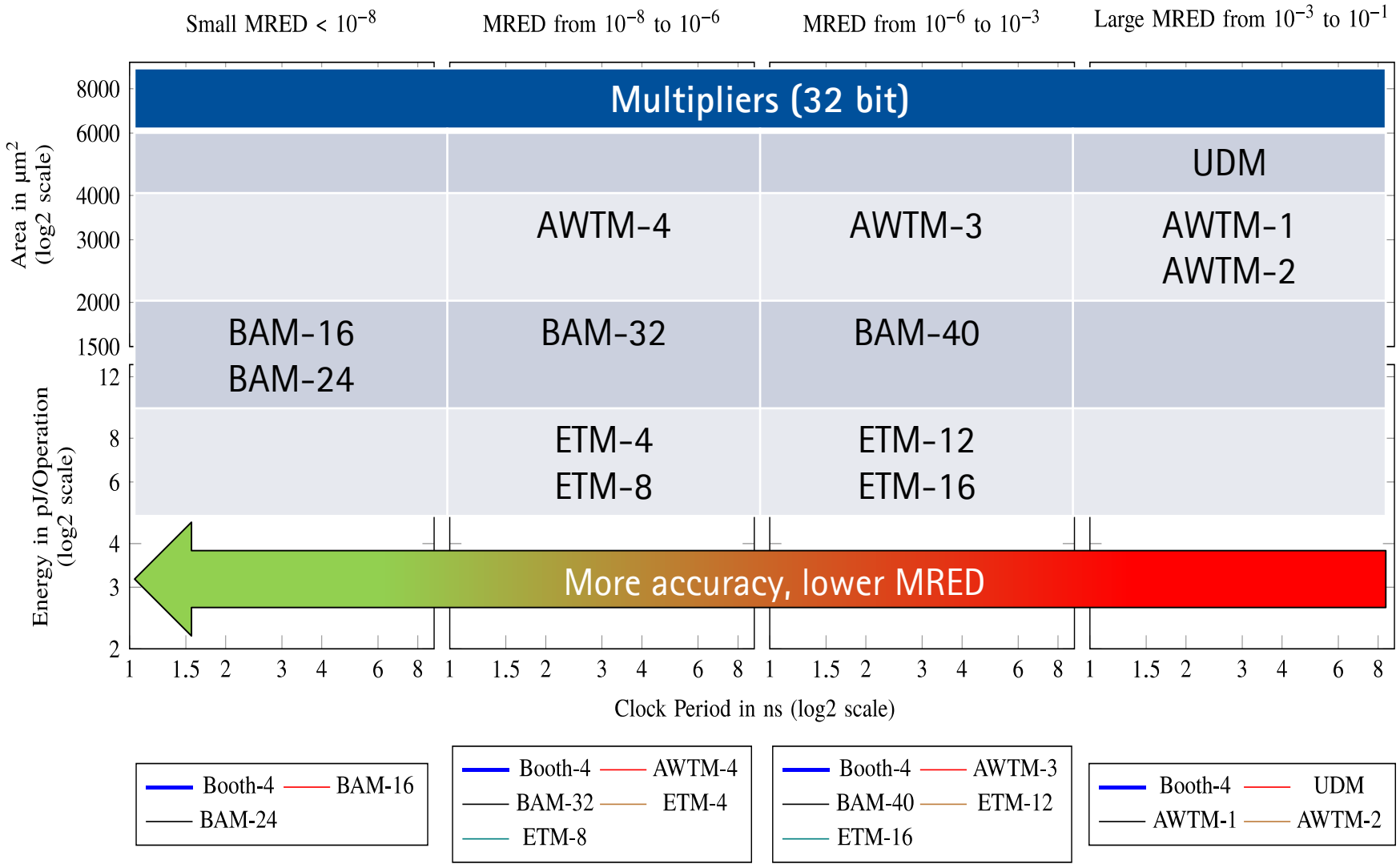
ATE-Accuracy Profiling: Approximate Adders



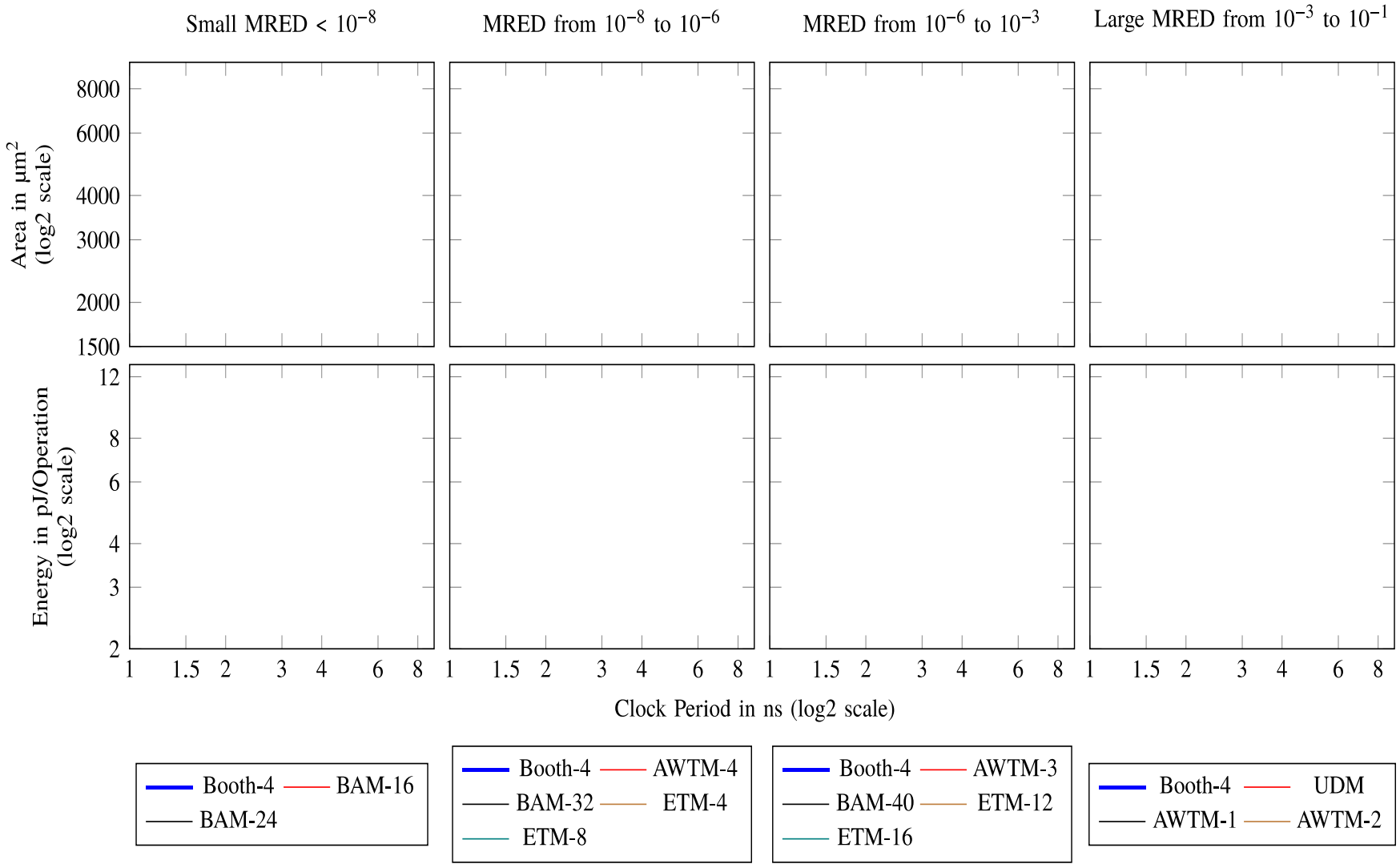
ATE-Accuracy Profiling: Approximate Multipliers



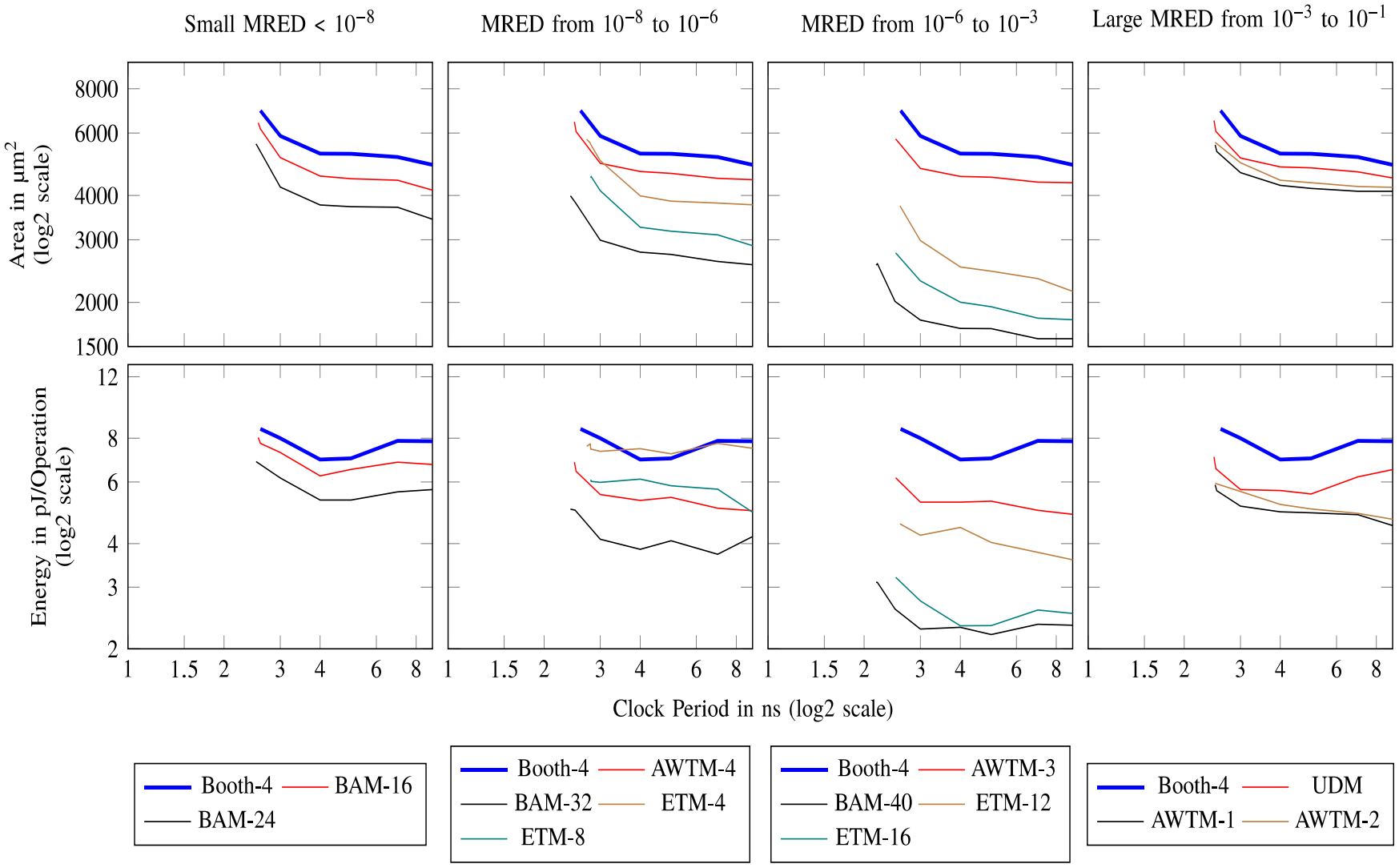
ATE-Accuracy Profiling: Approximate Multipliers



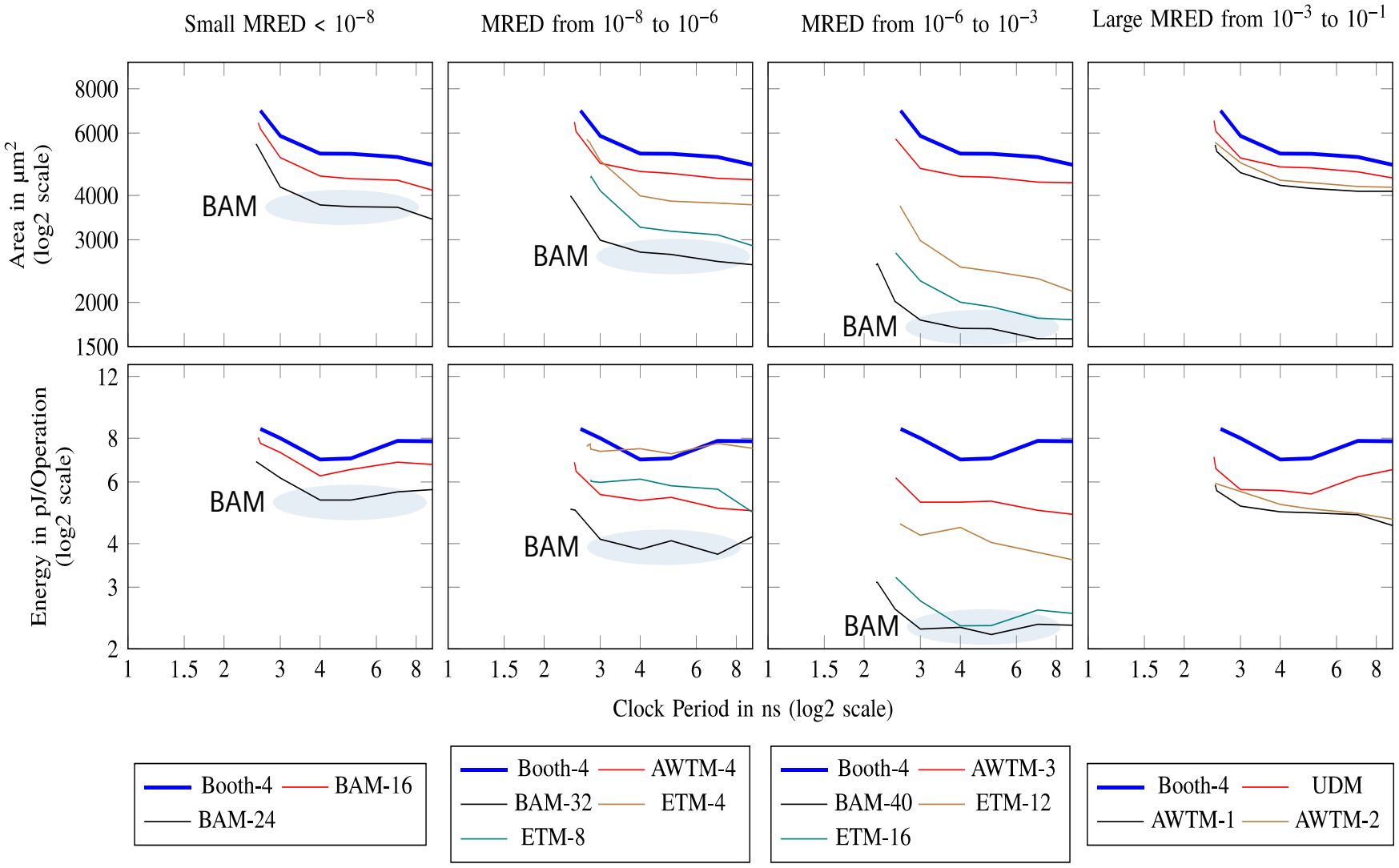
ATE-Accuracy Profiling: Approximate Multipliers



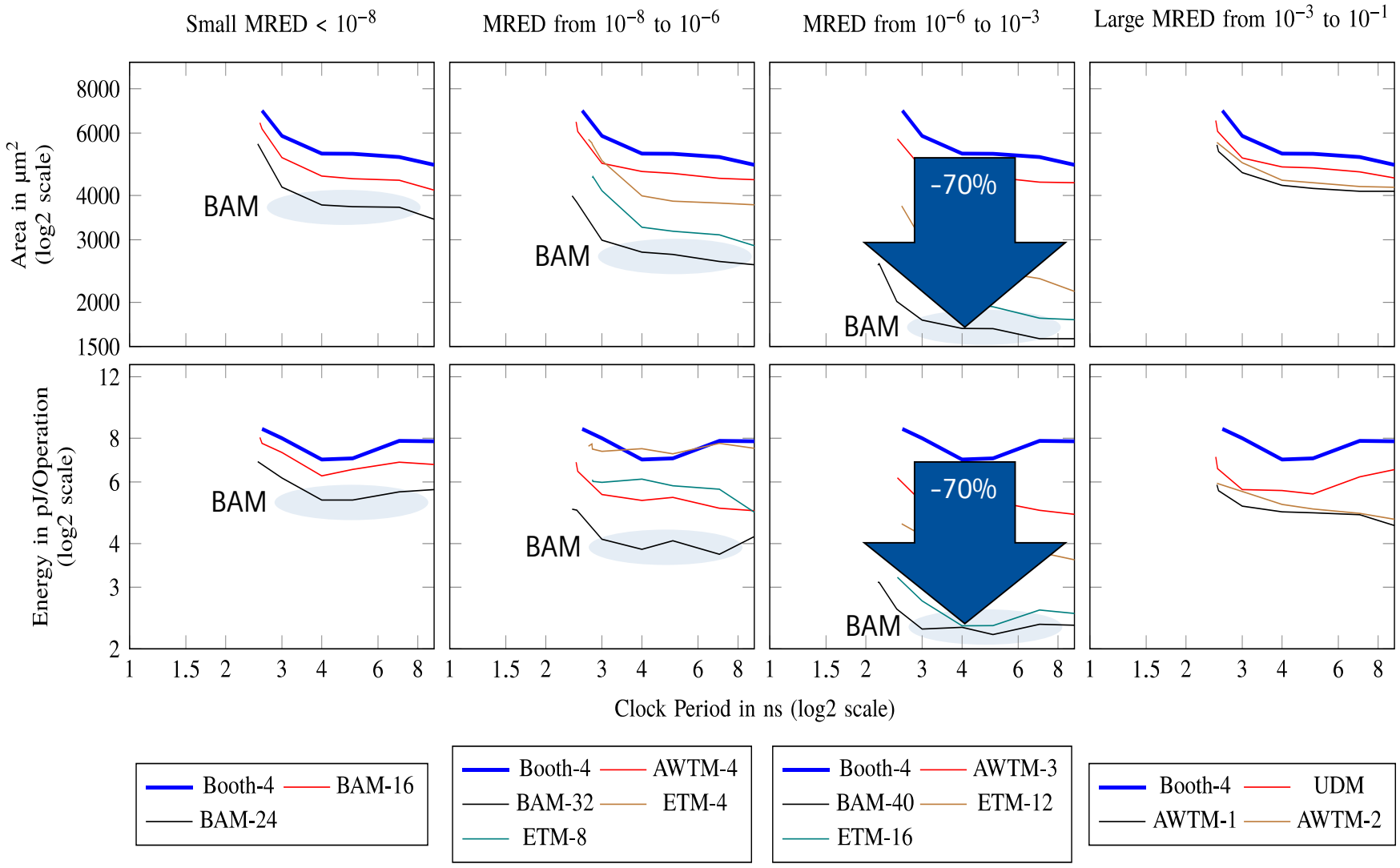
ATE-Accuracy Profiling: Approximate Multipliers



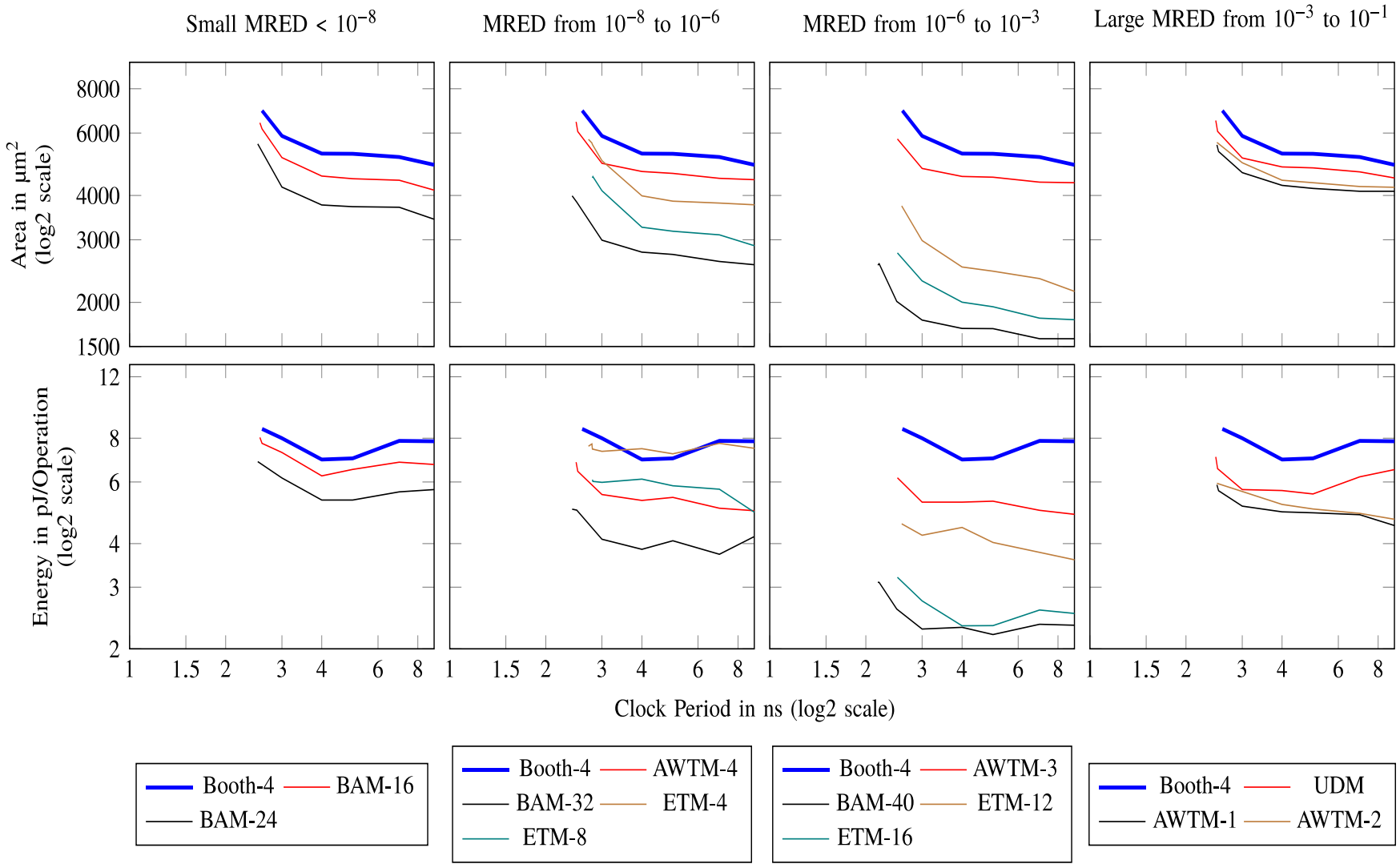
ATE-Accuracy Profiling: Approximate Multipliers



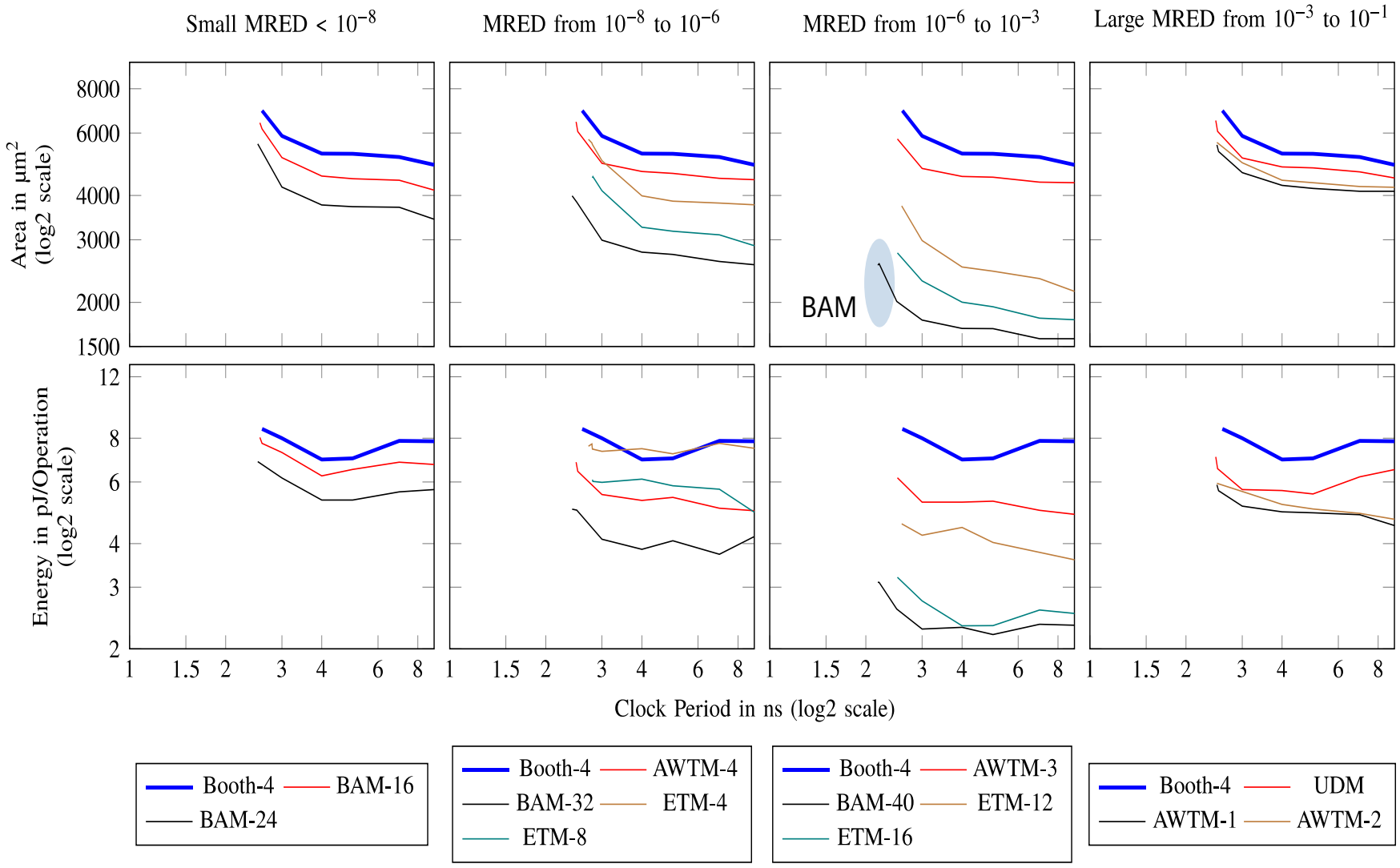
ATE-Accuracy Profiling: Approximate Multipliers



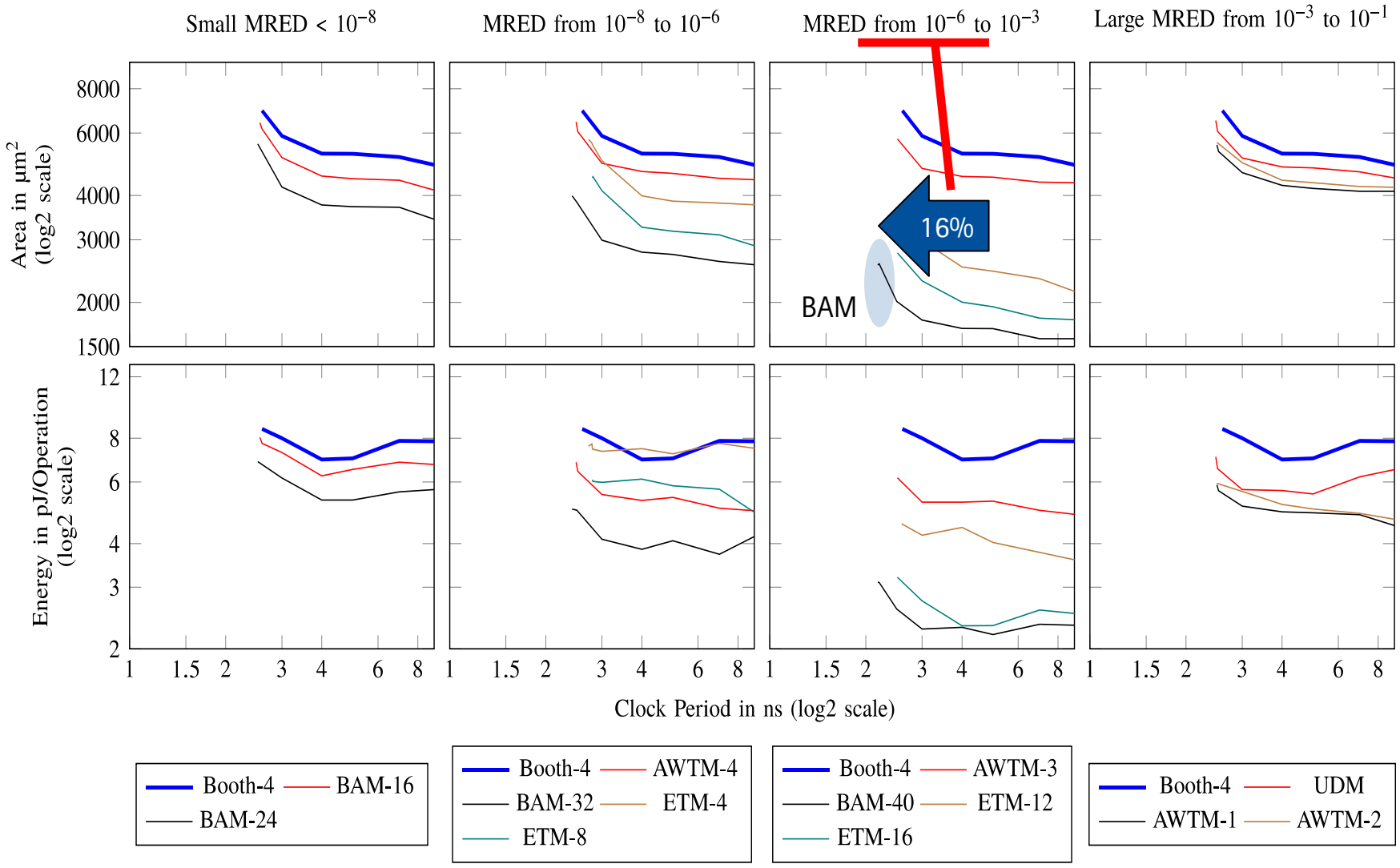
ATE-Accuracy Profiling: Approximate Multipliers



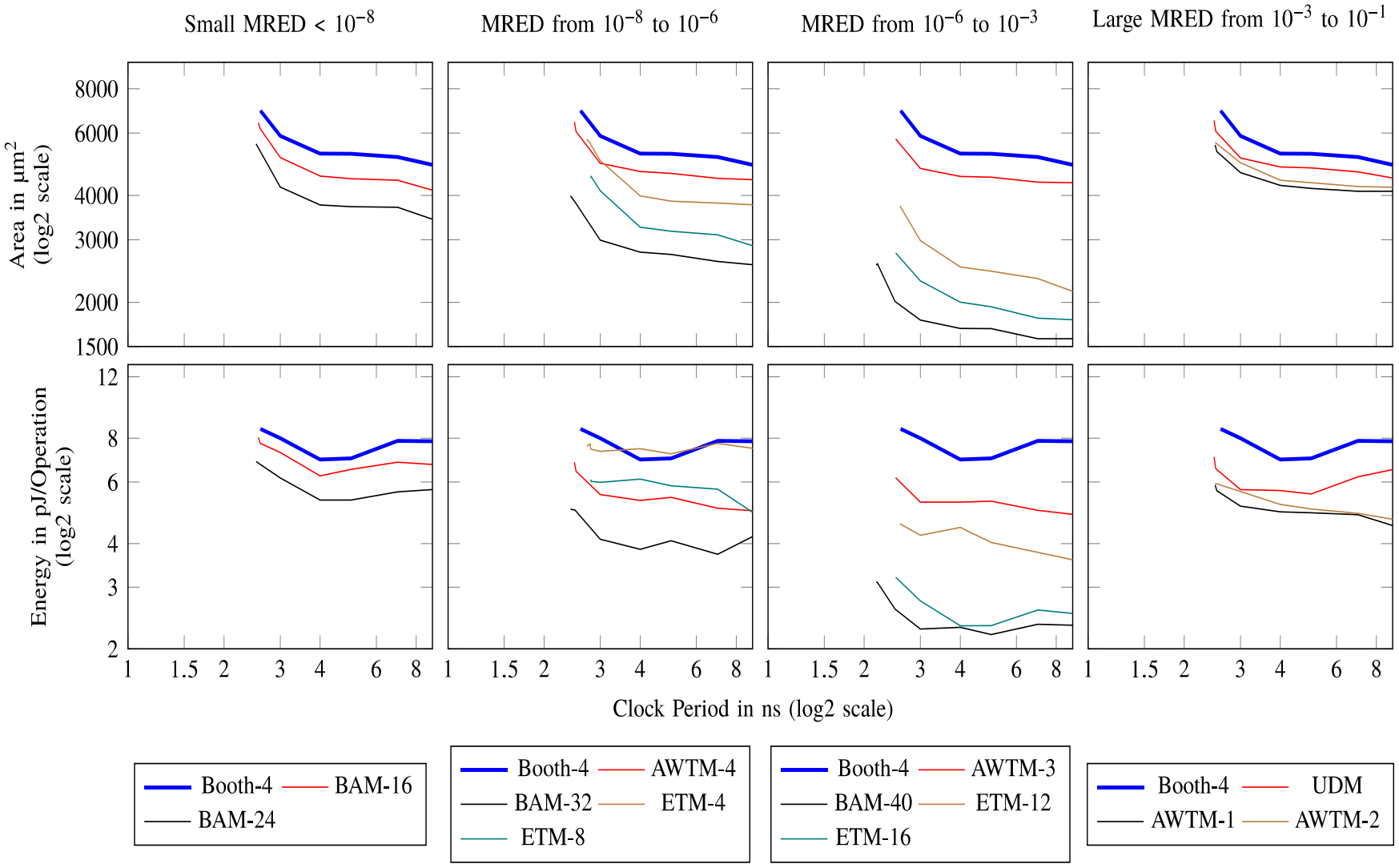
ATE-Accuracy Profiling: Approximate Multipliers



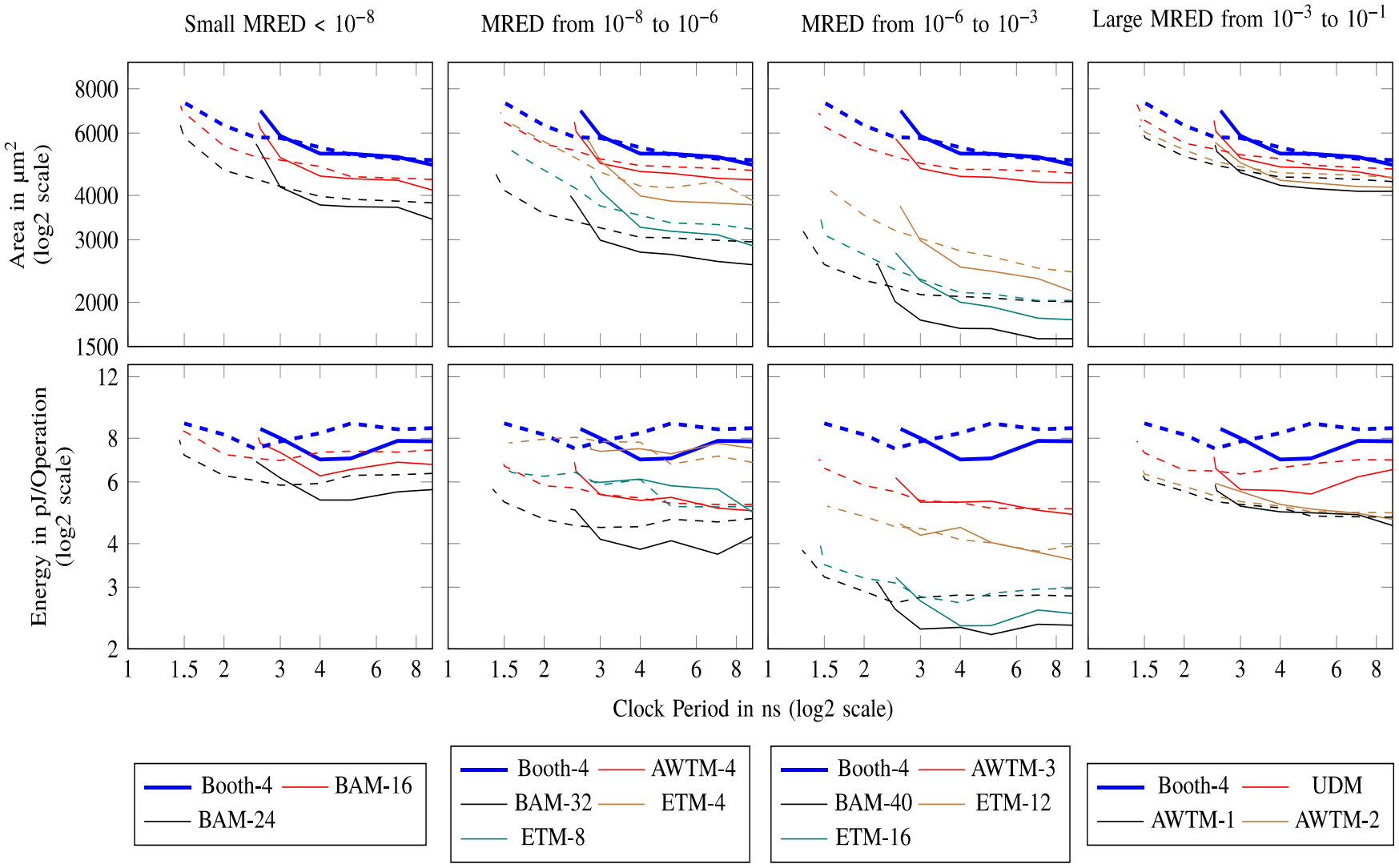
ATE-Accuracy Profiling: Approximate Multipliers



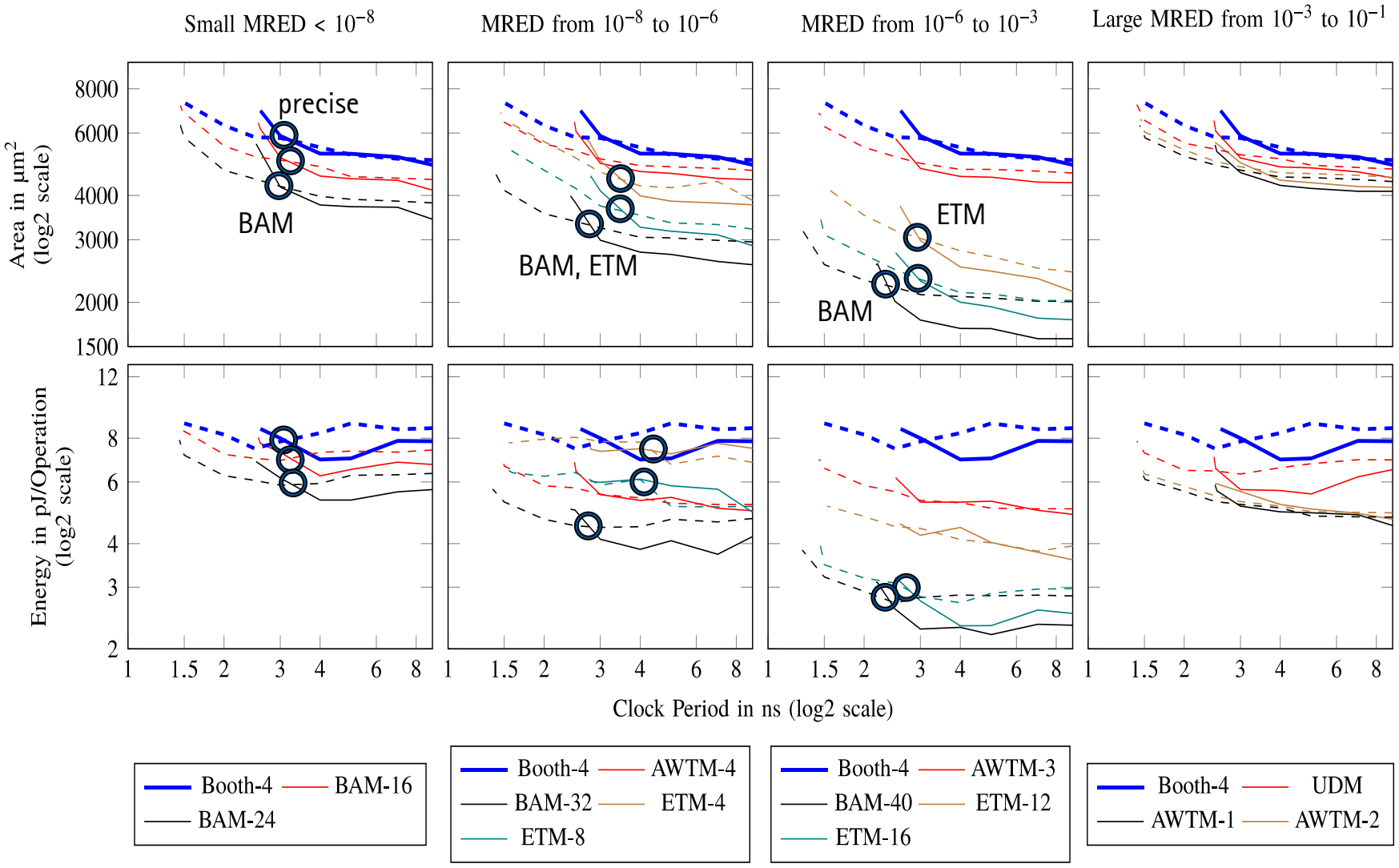
ATE-Accuracy Profiling: Approximate Multipliers



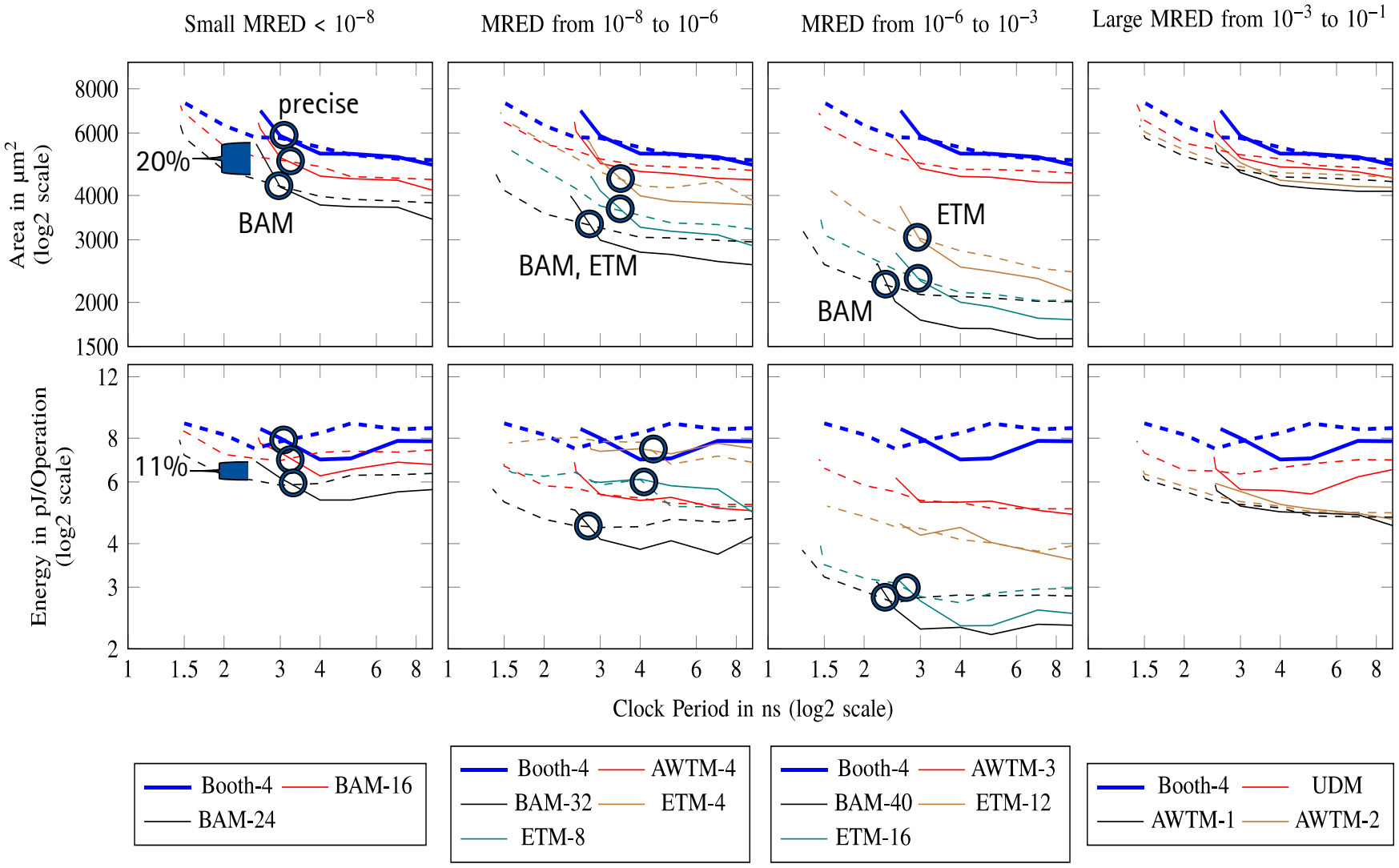
ATE-Accuracy Profiling: Approximate Multipliers



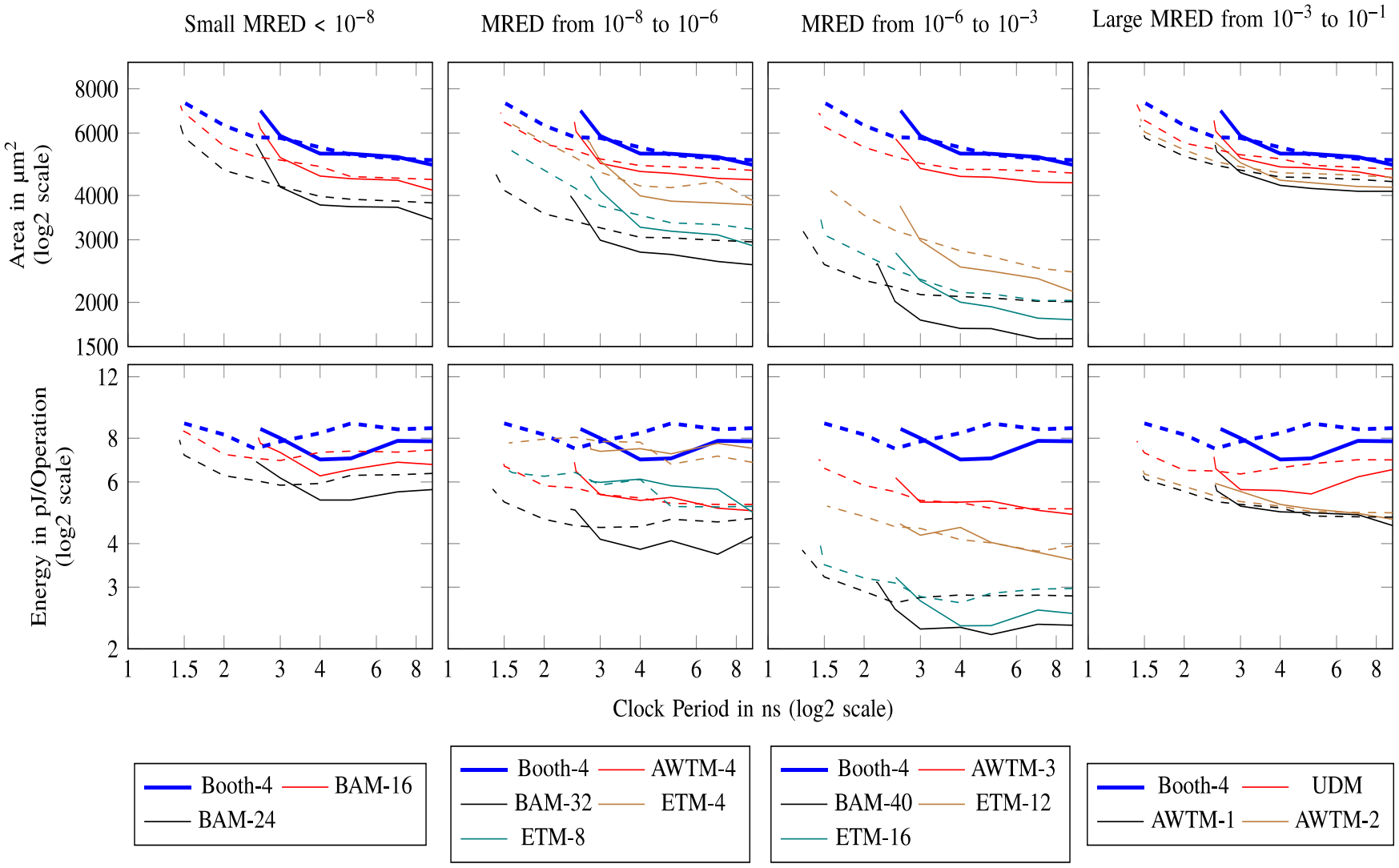
ATE-Accuracy Profiling: Approximate Multipliers



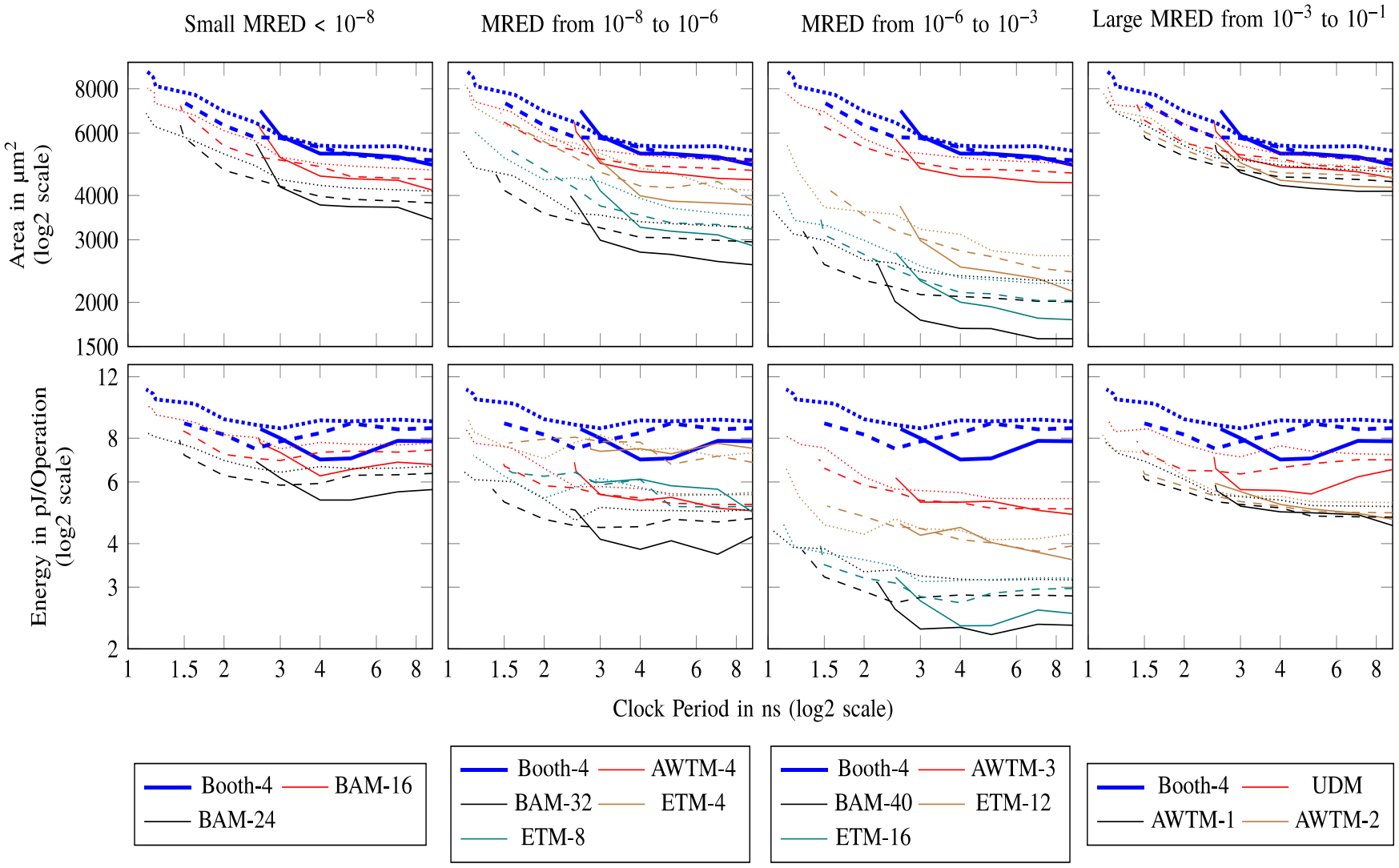
ATE-Accuracy Profiling: Approximate Multipliers



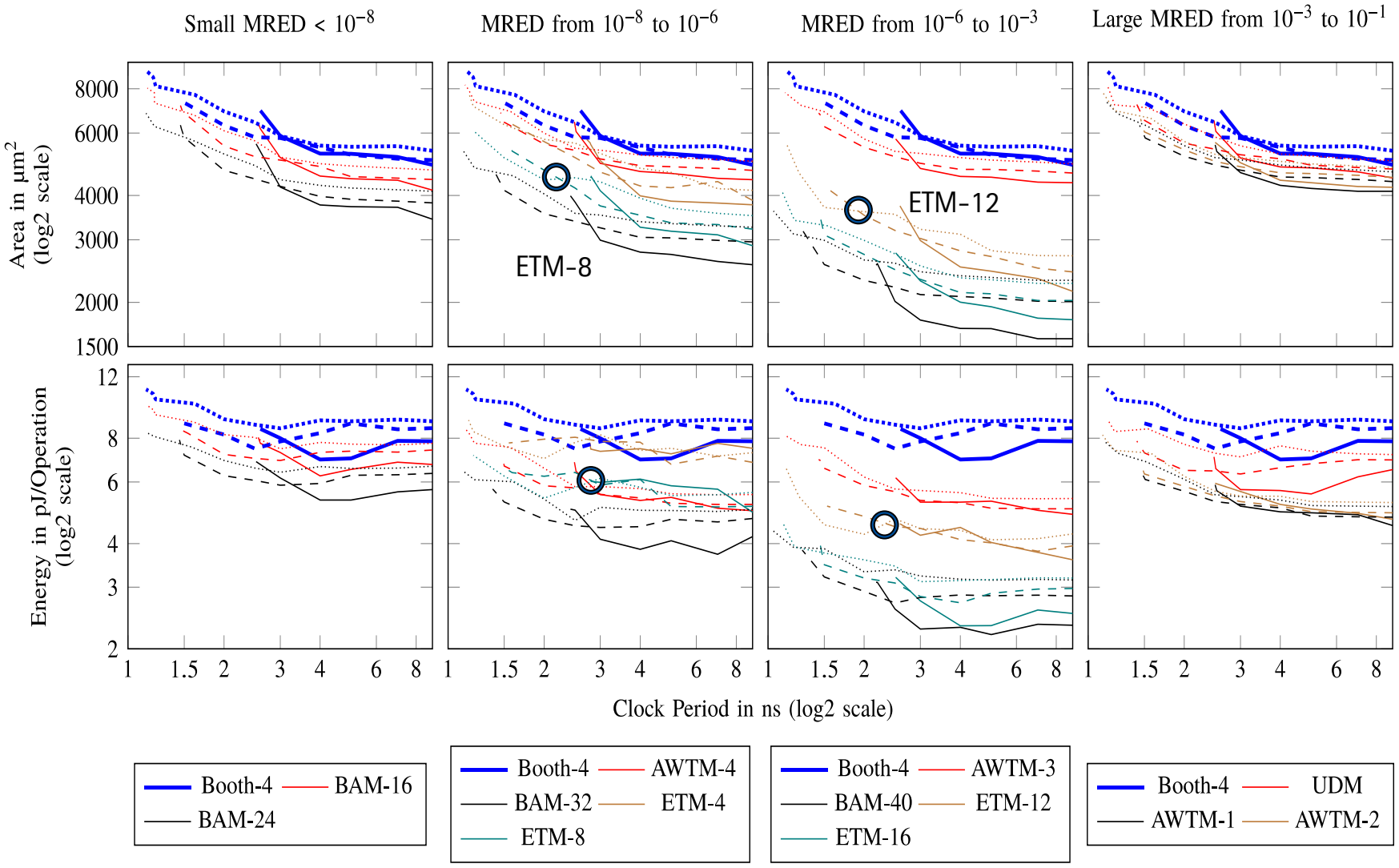
ATE-Accuracy Profiling: Approximate Multipliers



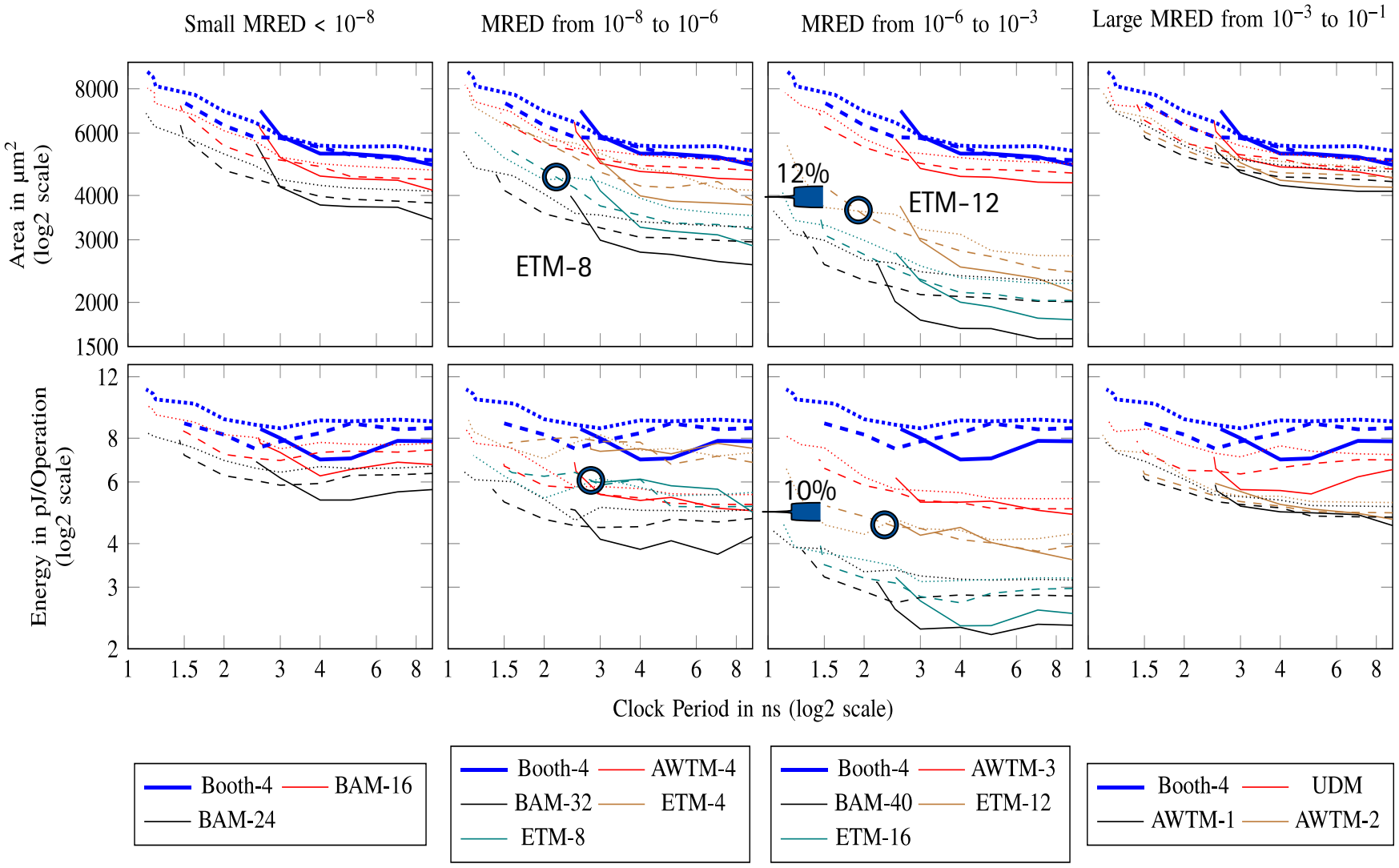
ATE-Accuracy Profiling: Approximate Multipliers



ATE-Accuracy Profiling: Approximate Multipliers



ATE-Accuracy Profiling: Approximate Multipliers



Conclusion

- Exploration of approximate adder and multipliers with a generic design and parameterizable amount of pipeline stages
  - Pipelining applied by register balancing

Conclusion

- Exploration of approximate adder and multipliers with a generic design and parameterizable amount of pipeline stages
  - Pipelining applied by register balancing

- Besides to more performance, pipelining can increase area and energy efficiency of arithmetic units at a fixed performance
  - Pipelining-aware two-phase synthesis flow
  - Area reduction of up to 20%
  - Energy reduction of up to 11%

Conclusion

- Exploration of approximate adder and multipliers with a generic design and parameterizable amount of pipeline stages
  - Pipelining applied by register balancing

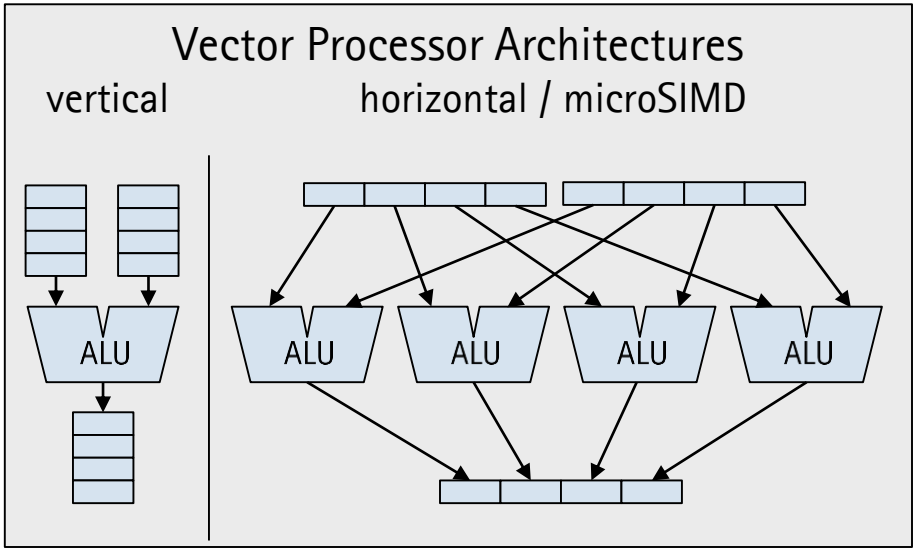
- Besides to more performance, pipelining can increase area and energy efficiency of arithmetic units at a fixed performance
  - Pipelining-aware two-phase synthesis flow
  - Area reduction of up to 20%
  - Energy reduction of up to 11%

Pipelining aids in meeting the target timing constraint without switching to an approximate unit with lower accuracy

Thank you for your attention!

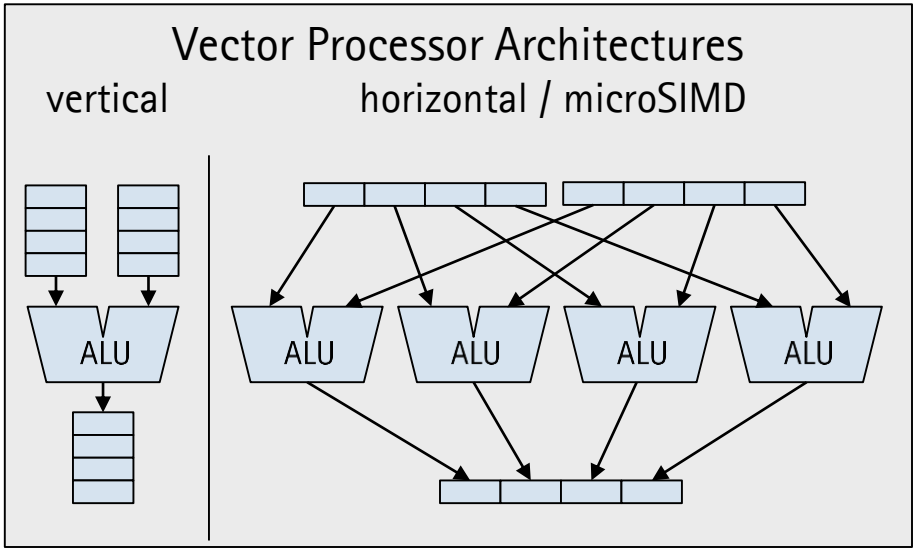
Backup: Vector Processor Architectures

- Specialized architectures for operating on independent, massively vectorizable data



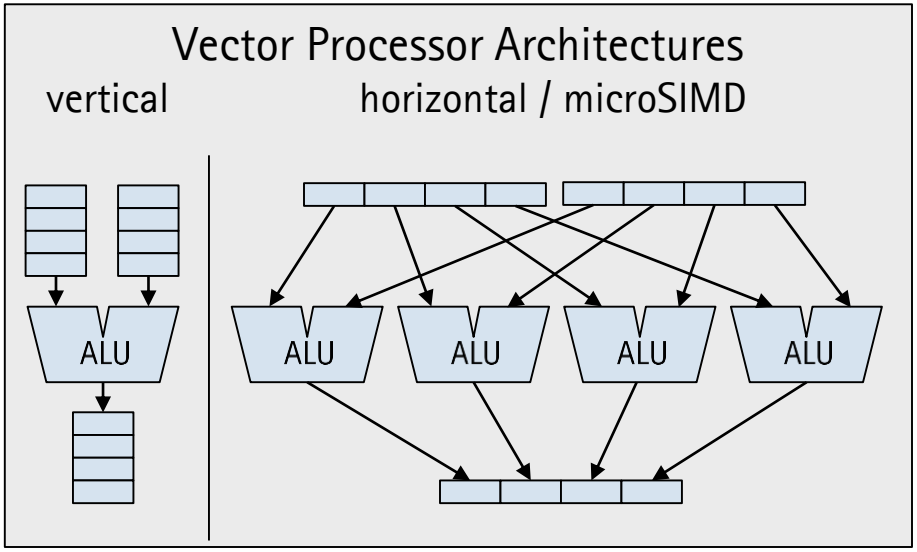
Backup: Vector Processor Architectures

- Specialized architectures for operating on independent, massively vectorizable data
  - Found in image processing, e.g., filtering (2D convolution, MAC), image differences (pixel-wise subtraction)



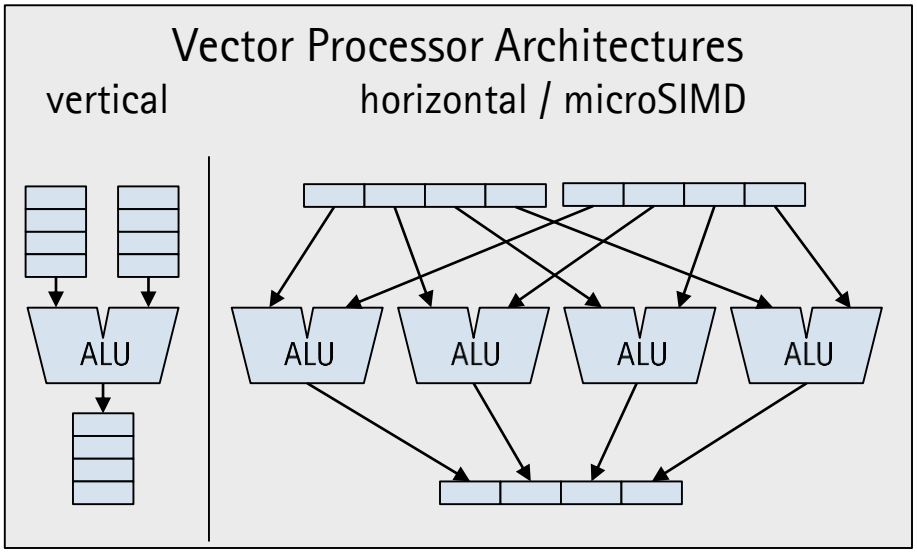
Backup: Vector Processor Architectures

- Specialized architectures for operating on independent, massively vectorizable data
  - Found in image processing, e.g., filtering (2D convolution, MAC), image differences (pixel-wise subtraction)
  - Programming flexibility while maintaining high processing performance

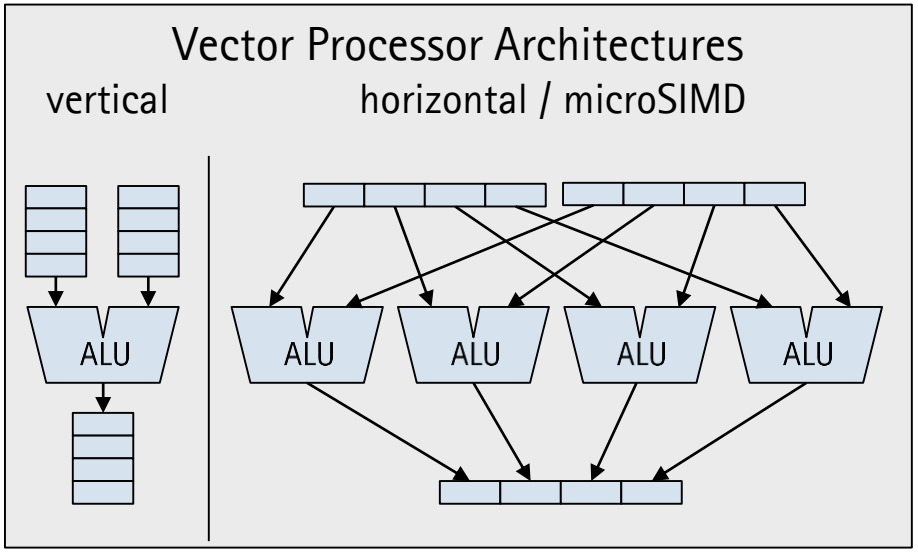


Backup: Vector Processor Architectures

- Specialized architectures for operating on independent, massively vectorizable data
  - Found in image processing, e.g., filtering (2D convolution, MAC), image differences (pixel-wise subtraction)
  - Programming flexibility while maintaining high processing performance
  - Approximate arithmetic for higher performance, area- or energy-efficiency
    - Approximate adder and multiplier designs

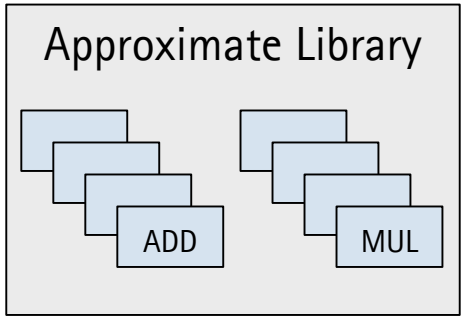


Backup: Analysis Framework for *Approximate* and *Stochastic Computing* Processor Architectures

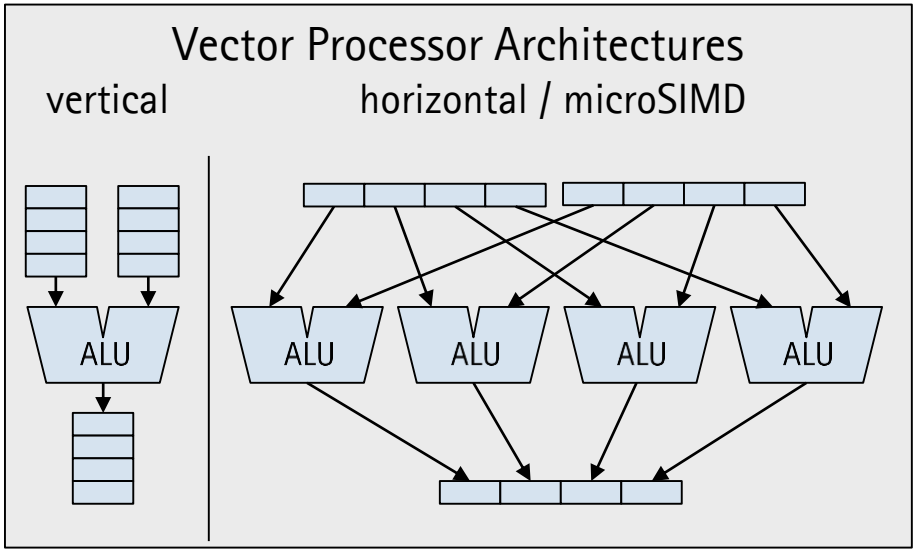


Generic VHDL description, exploit data-level parallelism of image processing algorithms

Backup: Analysis Framework for *Approximate* and *Stochastic Computing* Processor Architectures

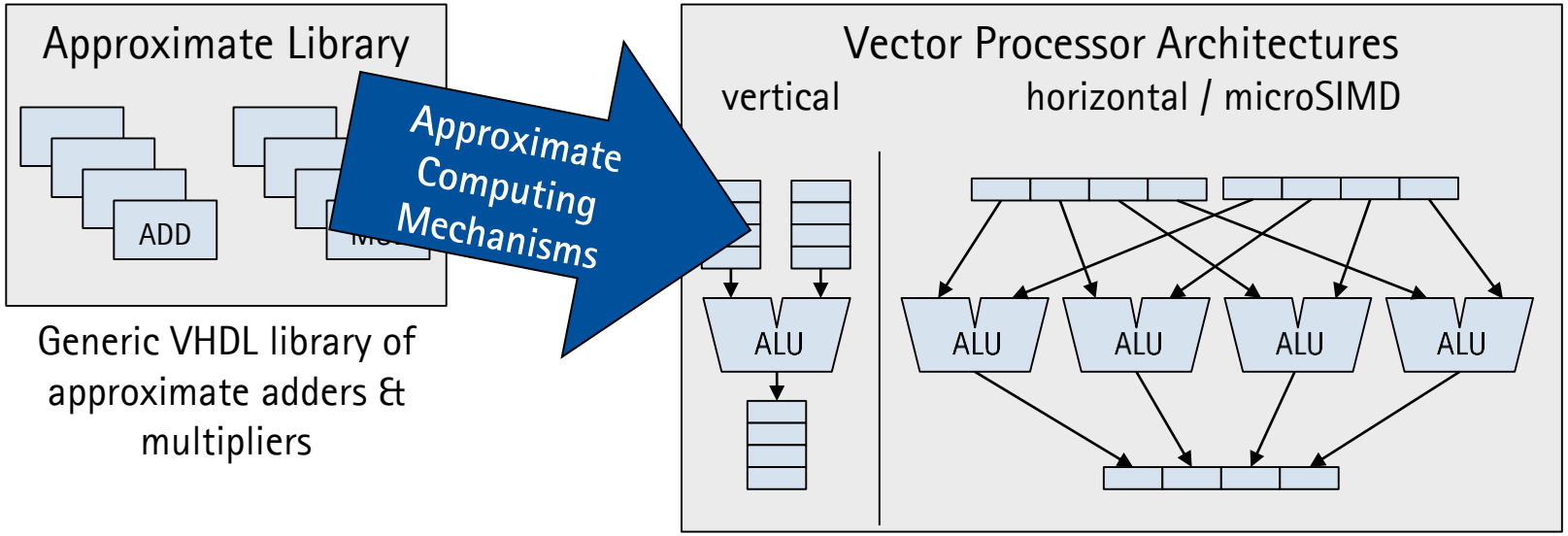


Generic VHDL library of approximate adders & multipliers



Generic VHDL description, exploit data-level parallelism of image processing algorithms

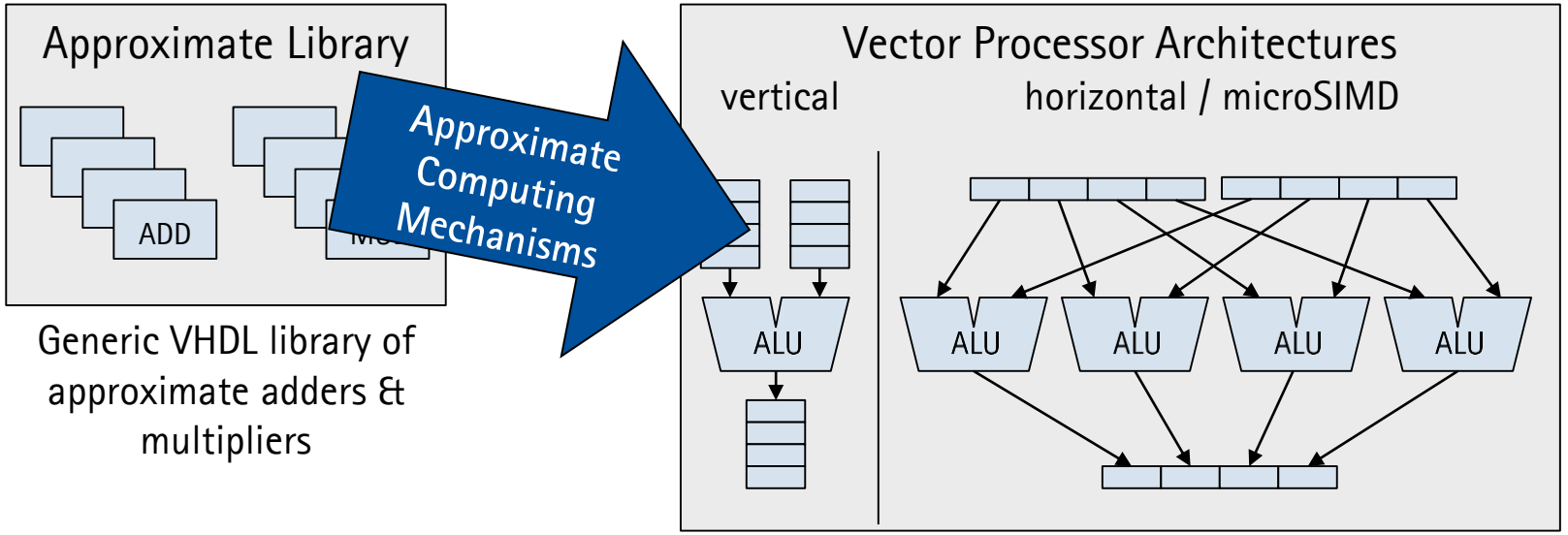
Backup: Analysis Framework for *Approximate* and *Stochastic Computing* Processor Architectures



Generic VHDL library of approximate adders & multipliers

Generic VHDL description, exploit data-level parallelism of image processing algorithms

Backup: Analysis Framework for *Approximate* and *Stochastic Computing* Processor Architectures

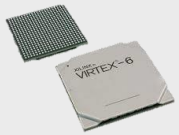


Generic VHDL library of approximate adders & multipliers

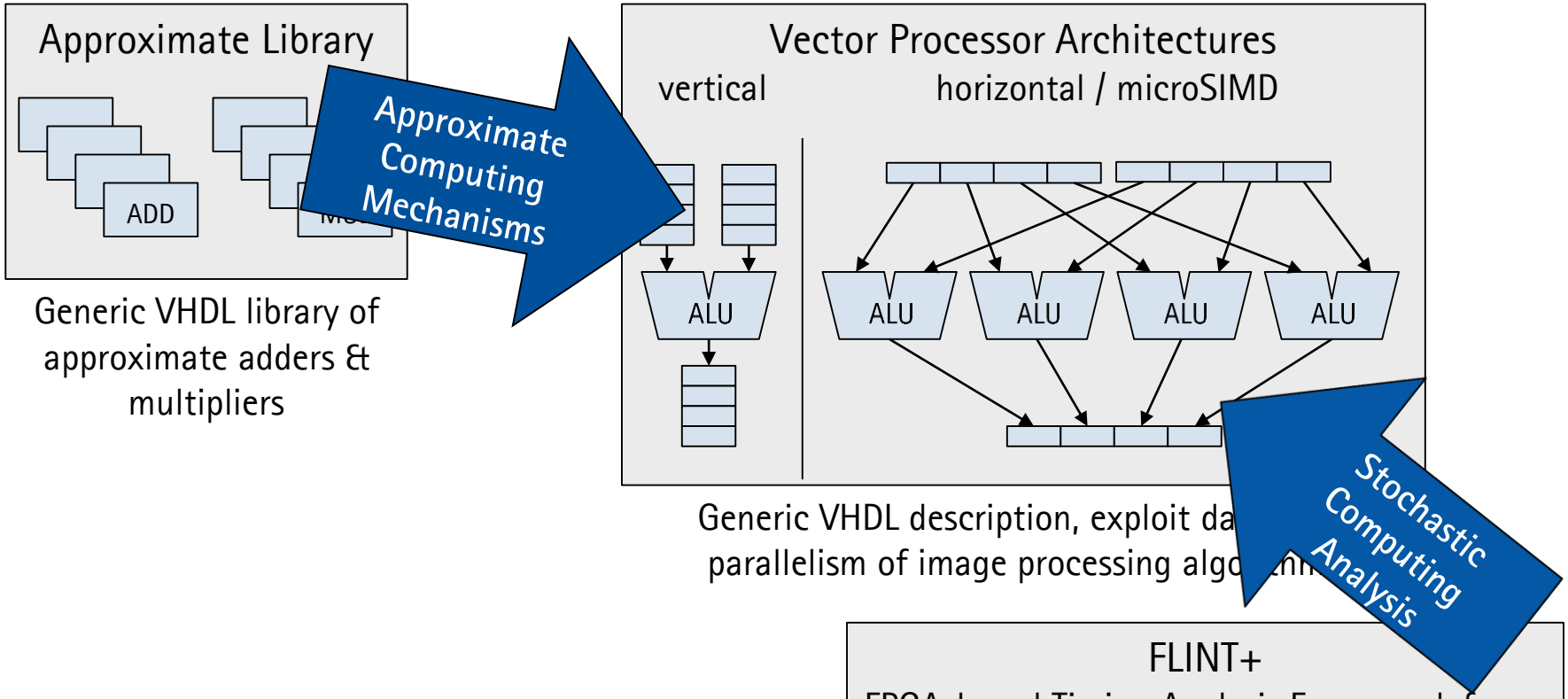
Generic VHDL description, exploit data-level parallelism of image processing algorithms

FLINT+

FPGA-based Timing Analysis Framework for Stochastic Computing Operation



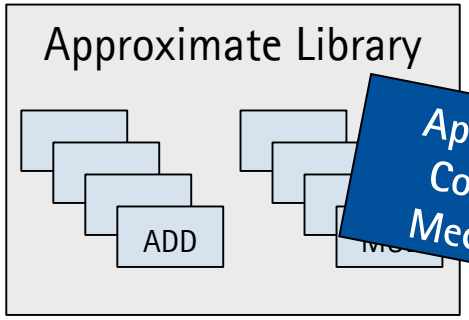
Backup: Analysis Framework for *Approximate* and *Stochastic Computing* Processor Architectures



Generic VHDL library of approximate adders & multipliers

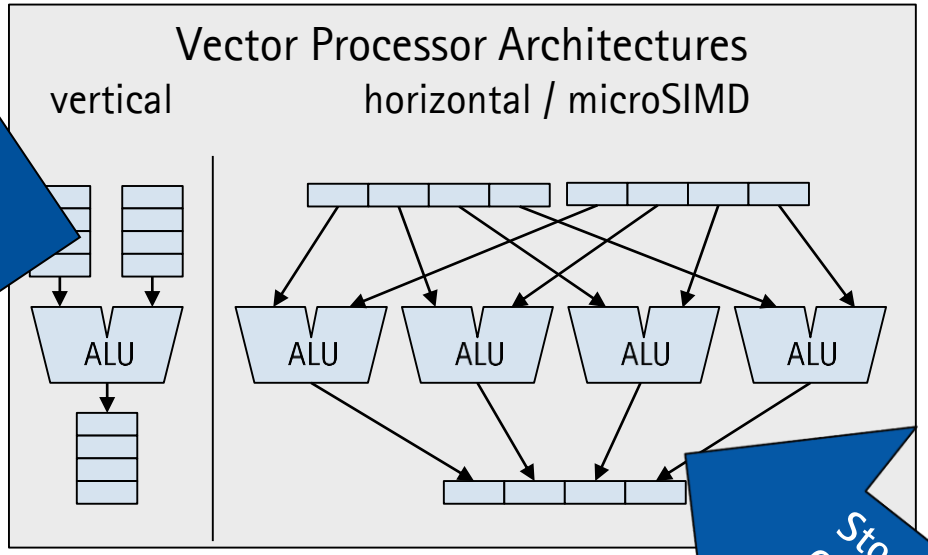
Generic VHDL description, exploit da
parallelism of image processing algo

Backup: Analysis Framework for *Approximate* and *Stochastic Computing* Processor Architectures



Generic VHDL library of approximate adders & multipliers

Approximate Computing Mechanisms

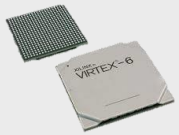


Generic VHDL description, exploit da
parallelism of image processing algo

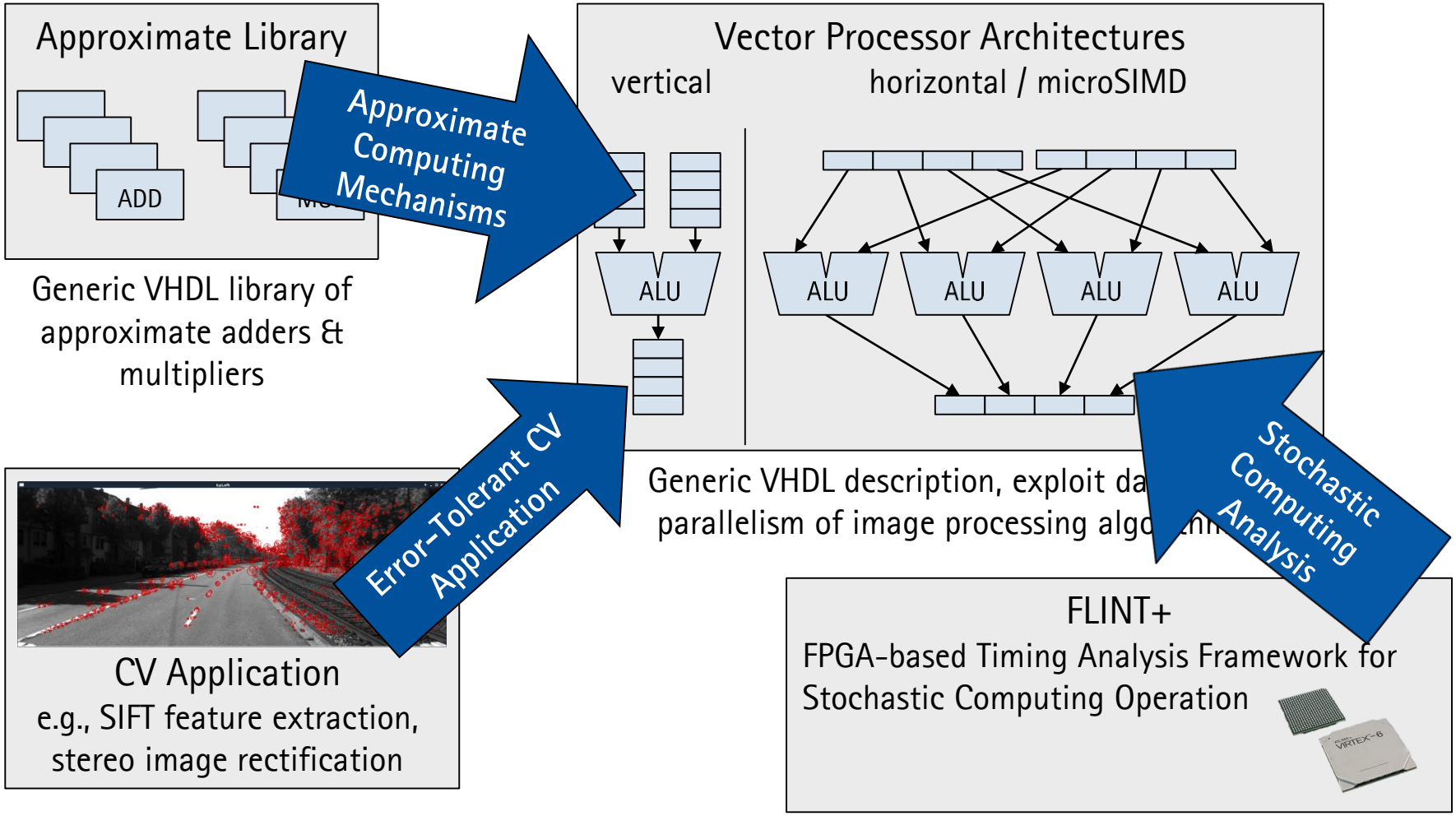
Stochastic Computing Analysis

CV Application
e.g., SIFT feature extraction, stereo image rectification

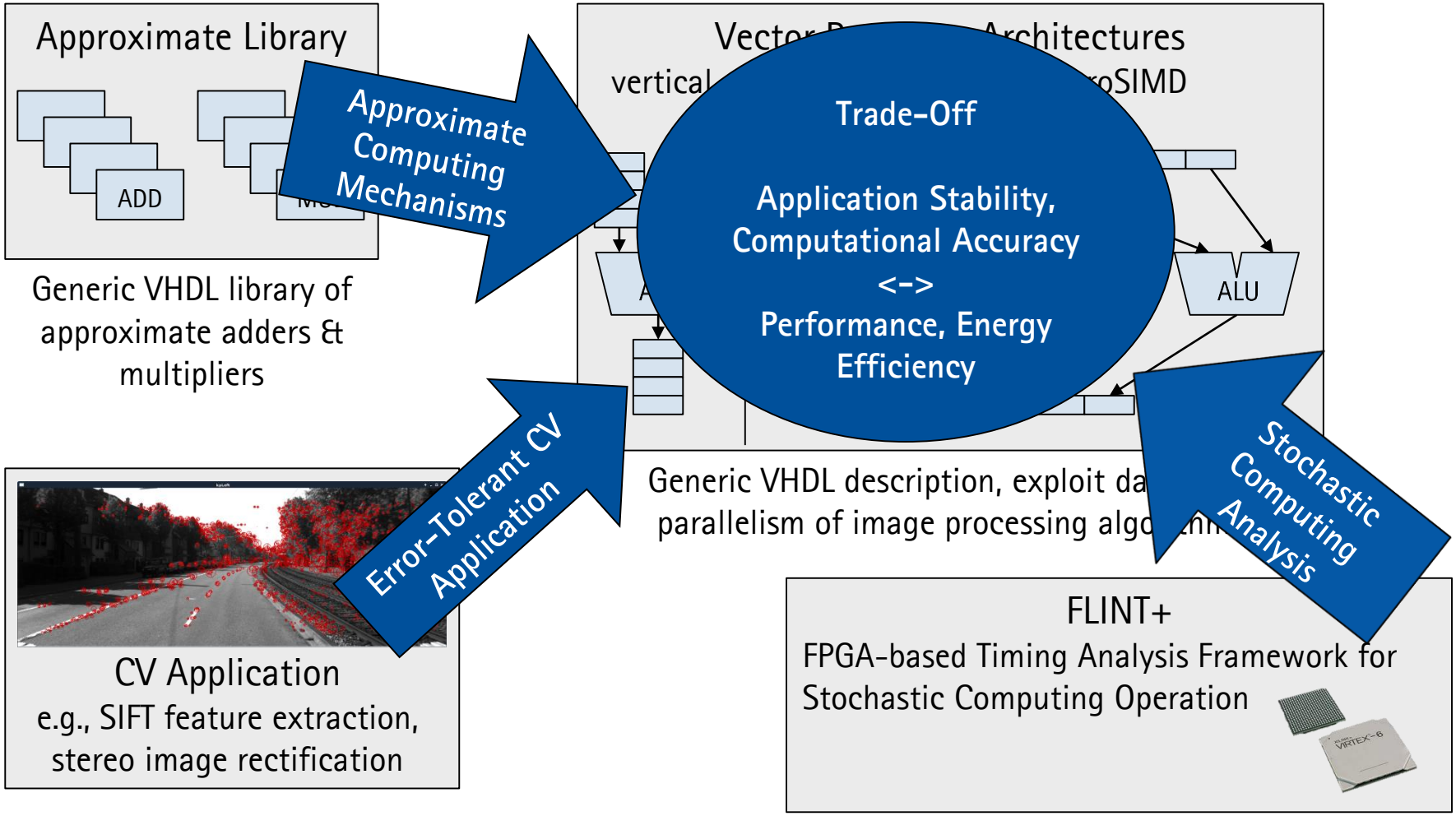
FLINT+
FPGA-based Timing Analysis Framework for Stochastic Computing Operation



Backup: Analysis Framework for *Approximate* and *Stochastic Computing* Processor Architectures



Backup: Analysis Framework for *Approximate* and *Stochastic Computing* Processor Architectures



Outline

Outline

- Generic VHDL implementation strategy for approximate adder and multiplier designs
  - Inferring optimized precise sub-components

Outline

- Generic VHDL implementation strategy for approximate adder and multiplier designs
  - Inferring optimized precise sub-components
- Pipelining-aware ASIC synthesis flow for area-efficient gate-level implementations
  - Configurable number of pipeline stages

Outline

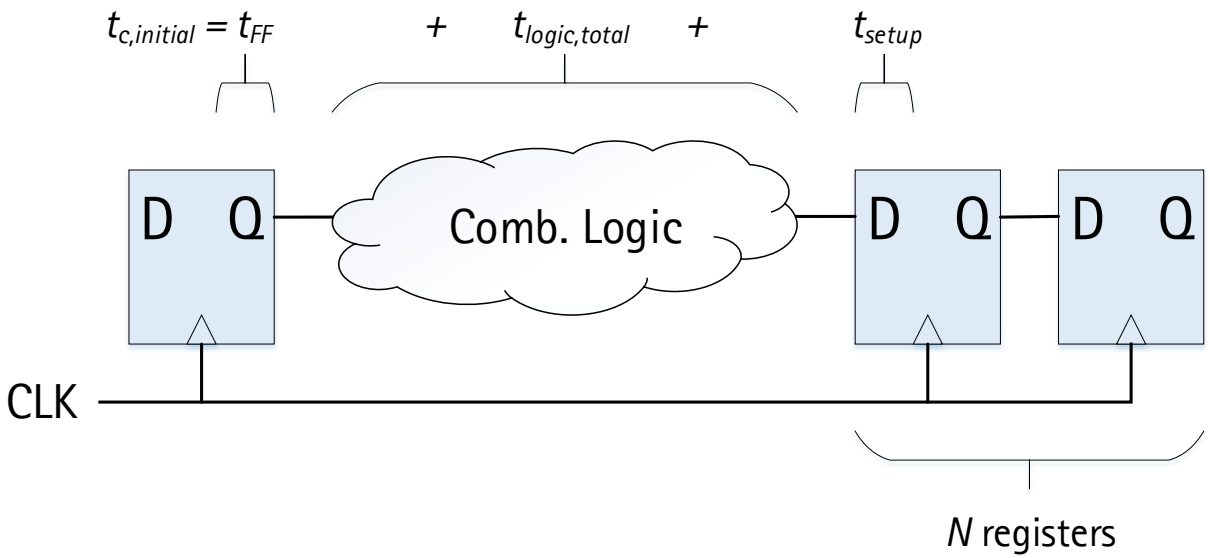
- Generic VHDL implementation strategy for approximate adder and multiplier designs
  - Inferring optimized precise sub-components
- Pipelining-aware ASIC synthesis flow for area-efficient gate-level implementations
  - Configurable number of pipeline stages
- Area-Timing-Energy-Accuracy profiling for pipelined approximate adders and multipliers

Two-Phase Pipelining-Aware ASIC Synthesis Flow

1. Architecture selection and mapping with a *relaxed* timing constraint → more area-efficient

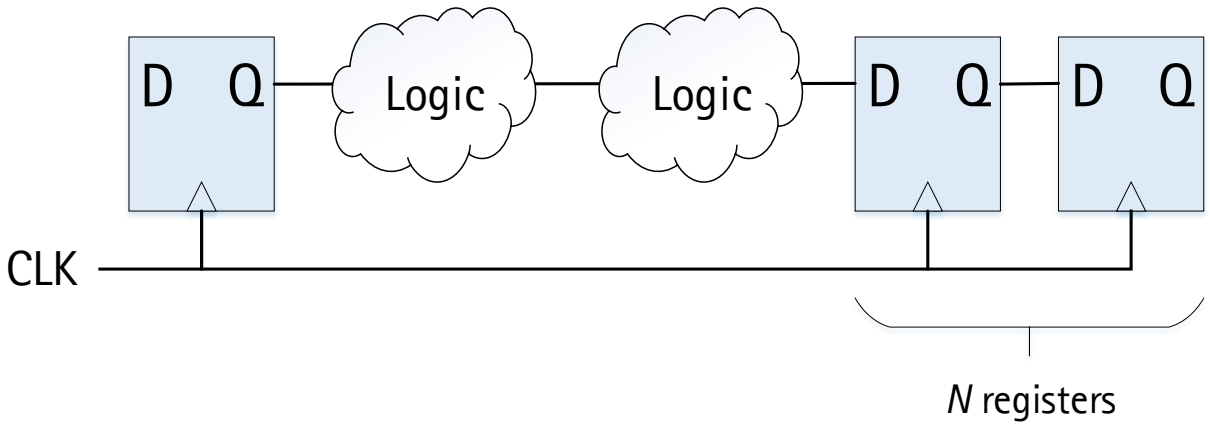
Two-Phase Pipelining-Aware ASIC Synthesis Flow

1. Architecture selection and mapping with a *relaxed* timing constraint → more area-efficient



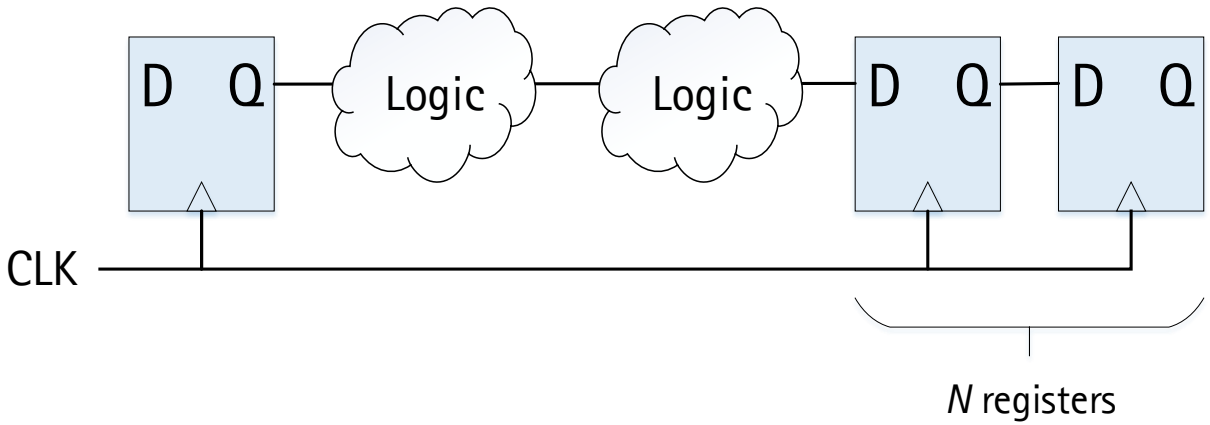
Two-Phase Pipelining-Aware ASIC Synthesis Flow

1. Architecture selection and mapping with a *relaxed* timing constraint → more area-efficient



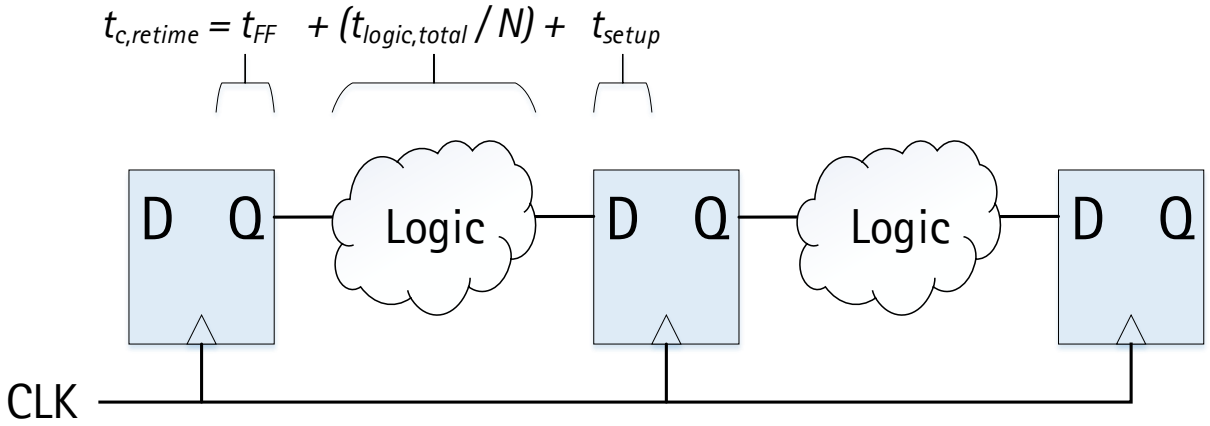
Two-Phase Pipelining-Aware ASIC Synthesis Flow

1. Architecture selection and mapping with a *relaxed* timing constraint → more area-efficient
2. Incremental synthesis with retiming/register balancing at the *desired* timing constraint



Two-Phase Pipelining-Aware ASIC Synthesis Flow

1. Architecture selection and mapping with a *relaxed* timing constraint → more area-efficient
2. Incremental synthesis with retiming/register balancing at the *desired* timing constraint



Two-Phase Pipelining-Aware ASIC Synthesis Flow

1. Architecture selection and mapping with a *relaxed* timing constraint → more area-efficient
2. Incremental synthesis with retiming/register balancing at the *desired* timing constraint