An Algorithm for Inversion in $\mathrm{GF}(2^m)$ Suitable for Implementation Using a Polynomial Multiply Instruction on $\mathrm{GF}(2)$

K. Kobayashi, N. Takagi, and K. Takagi

Graduate School of Information Science, Nagoya University

Outline

- Background and objective
- Preliminaries
  - $\mathrm{GF}(2^m)$
  - A polynomial multiply instruction on $\mathrm{GF}(2)$
  - A conventional algorithm for inversion in $\mathrm{GF}(2^m)$
- A new algorithm for inversion in $\mathrm{GF}(2^m)$
- Evaluation
- Concluding remarks

Background and Objective

- $\mathrm{GF}(2^m)$
  - plays important roles in error-correcting codes and cryptography
  - A fast algorithm for inversion in $\mathrm{GF}(2^m)$ is required
- Polynomial multiply instruction on $\mathrm{GF}(2)$
  - accelerates multiplication in $\mathrm{GF}(2^m)$.

We propose a fast algorithm for inversion in $\mathrm{GF}(2^m)$
that is suitable for implementation
using a polynomial multiply instruction on $\mathrm{GF}(2)$

$\mathrm{GF}(2^m)$ **(1/2)**

- $\mathrm{GF}(2^m)$
  - extension field of $\mathrm{GF}(2)$
  - any element $A(x) \in \mathrm{GF}(2^m)$
    - $A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 \quad (a_i \in \{0, 1\})$
- Addition in $\mathrm{GF}(2^m)$
  - polynomial addition on $\mathrm{GF}(2)$
  - $A(x) + B(x)$
    $= ((a_{m-1} + b_{m-1}) \bmod 2)x^{m-1} + \cdots + ((a_0 + b_0) \bmod 2)$
  - executed by exclusive-OR operation for every coefficient

$\mathrm{GF}(2^m)$ **(2/2)**

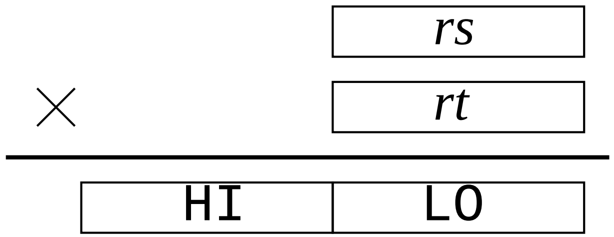
- Multiplication in $\mathrm{GF}(2^m)$
  - polynomial multiplication modulo $G(x)$ on $\mathrm{GF}(2)$
    - $G(x)$: the irreducible polynomial with degree m
  - $A(x) \cdot B(x) = A(x) \times B(x) \bmod G(x)$
    - $\cdot$: multiplication in $\mathrm{GF}(2^m)$
    - $\times$: polynomial multiplication in $\mathrm{GF}(2)$
- Multiplicative inverse of $A(x)$
  - The element $A^{-1}(x)$ is such that

$$A(x) \cdot A^{-1}(x) = 1.$$

  - time-consuming operation

MULGF2

- MULGF2 instruction
  - A typical polynomial multiply instruction on $\mathrm{GF}(2)$
  - calculates the 2-word polynomial product from two 1-word polynomial operands



  - accelerates multiplication in $\mathrm{GF}(2^m)$
  - A multiplier for MULGF2 can be realized very easily
    - “carry-free” version of an integer multiplier

Algorithm for Inversion in $\mathrm{GF}(2^m)$

- By extending the Euclid's algorithm for polynomial, we can execute inversion in $\mathrm{GF}(2^m)$.

$R_{-1}(x) := G(x);$
$R_0(x) := A(x);$
$j := 0;$
repeat
  $j := j + 1;$
  $Q_j(x) := R_{j-2}(x) \div R_{j-1}(x);$
  $R_j(x) := R_{j-2}(x) - Q_j(x) \times R_{j-1}(x);$

until $R_j(x) = 0;$
outputs $R_{j-1}(x)$ *as* $GCD(A(x), G(x))$

Algorithm for Inversion in $\mathrm{GF}(2^m)$

- By extending the Euclid's algorithm for polynomial, we can execute inversion in $\mathrm{GF}(2^m)$.

$R_{-1}(x) := G(x)$; $U_{-1}(x) := 0;$
$R_0(x) := A(x)$; $U_0(x) := 1;$
$j := 0;$
repeat
  $j := j + 1;$
  $Q_j(x) := R_{j-2}(x) \div R_{j-1}(x);$
  $R_j(x) := R_{j-2}(x) - Q_j(x) \times R_{j-1}(x);$
  $U_j(x) := U_{j-2}(x) - Q_j(x) \times U_{j-1}(x);$
until $R_j(x) = 0;$
outputs $R_{j-1}(x)$ *as* $GCD(A(x), G(x))$
outputs $U_{j-1}(x)$ *as* $A^{-1}(x)$
($A(x) \times A^{-1}(x) \bmod G(x) = 1$)

Software Implementation of EA

- software implementation of the Euclid's algorithm

$S(x) := G(x); R(x) := A(x);$

while $R(x) \neq 0$ **do**

$\quad \delta := \deg(S(x)) - \deg(R(x));$

$\quad$ **if** $\deg(S(x)) < \deg(R(x))$ **then**

$\quad\quad R(x) \leftrightarrow S(x);\ \delta := -\delta;$

$\quad$ **end if**

$\quad S(x) := S(x) - x^{\delta} \times R(x);$

end while

1st iteration

S:	x^3	+	x^2	+	1
R:			x^2	+	1

Software Implementation of EA

- software implementation of the Euclid's algorithm

$S(x) := G(x); R(x) := A(x);$

while $R(x) \neq 0$ **do**

$\boxed{\delta := \deg(S(x)) - \deg(R(x));}$

if $\deg(S(x)) < \deg(R(x))$ **then**

$R(x) \leftrightarrow S(x);\ \delta := -\delta;$

end if

$\boxed{S(x) := S(x) - x^{\delta} \times R(x);}$

end while

1st iteration

S:	x^{3}	+	x^{2}	+	1
R:			x^{2}	+	1

$S(x) := S(x) - x^{3-2} \times R(x);$

Software Implementation of EA

- software implementation of the Euclid's algorithm

$S(x) := G(x); R(x) := A(x);$

while $R(x) \neq 0$ **do**

  $\delta := \deg(S(x)) - \deg(R(x));$

  if $\deg(S(x)) < \deg(R(x))$ **then**

    $R(x) \leftrightarrow S(x);\ \ \delta := -\delta;$

  end if

  $S(x) := S(x) - x^{\delta} \times R(x);$

end while

1st iteration

S:	x^3	+	x^2		+	1
R:			x^2		+	1

2nd iteration

S:			x^2	+ x	+	1
R:			x^2		+	1

Software Implementation of EA

- software implementation of the Euclid's algorithm

$S(x) := G(x); R(x) := A(x);$

while $R(x) \neq 0$ **do**

$\boxed{\delta := \deg(S(x)) - \deg(R(x));}$

if $\deg(S(x)) < \deg(R(x))$ **then**

$R(x) \leftrightarrow S(x);\ \delta := -\delta;$

end if

$\boxed{S(x) := S(x) - x^{\delta} \times R(x)}$

end while

1st iteration

$S:\ x^3 + x^2 + 1$

$R:\ x^2 + 1$

2nd iteration

$S:\ x^2 + x + 1$

$R:\ x^2 + 1$

$S(x) := S(x) - x^{2-2} \times R(x);$

Software Implementation of EA

- software implementation of the Euclid's algorithm

$S(x) := G(x); R(x) := A(x);$

while $R(x) \neq 0$ **do**

 $\delta := \deg(S(x)) - \deg(R(x));$

 if $\deg(S(x)) < \deg(R(x))$ **then**

  $R(x) \leftrightarrow S(x);\ \delta := -\delta;$

 end if

 $S(x) := S(x) - x^{\delta} \times R(x);$

end while

1st iteration

S: $x^3 + x^2 + 1$

R: $x^2 + 1$

2nd iteration

S: $x^2 + x + 1$

R: $x^2 + 1$

3rd iteration

S: x^1

R: $x^2 + 1$

Software Implementation of EA

- software implementation of the Euclid's algorithm

$S(x) := G(x); R(x) := A(x);$

while $R(x) \neq 0$ **do**

$\delta := \deg(S(x)) - \deg(R(x));$

if $\deg(S(x)) < \deg(R(x))$ **then**

$R(x) \leftrightarrow S(x);\ \delta := -\delta;$

end if

$S(x) := S(x) - x^{\delta} \times R(x);$

end while

1st iteration

$S:\ x^3 + x^2 + 1$

$R:\ x^2 + 1$

2nd iteration

$S:\ x^2 + x + 1$

$R:\ x^2 + 1$

3rd iteration

$S:\ x^1$

$R:\ x^2 + 1$

$S(x) <-> R(x);$

$S(x) := S(x) - x^{2-1} \times R(x);$

Software Implementation of EA

- software implementation of the Euclid's algorithm

$S(x) := G(x); R(x) := A(x);$

while $R(x) \neq 0$ **do**

  $\delta := \deg(S(x)) - \deg(R(x));$

  if $\deg(S(x)) < \deg(R(x))$ **then**

    $R(x) \leftrightarrow S(x);\ \delta := -\delta;$

  end if

  $S(x) := S(x) - x^{\delta} \times R(x);$

end while

1st & 2nd iterations correspond to one polynomial division

1st iteration

S: $x^3 + x^2 + 1$

R: $x^2 + 1$

2nd iteration

S: $x^2 + x + 1$

R: $x^2 + 1$

3rd iteration

S: x^1

R: $x^2 + 1$

4th iteration

S: x^1

R: $x^2 + 1$

Main Idea

- Key point
  - The conventional algorithm can not use MULGF2 efficiently
  - $S(x) := S(x) - \boxed{x^\delta} \times R(x);$
- New algorithm
  - based on Brunner's hardware algorithm for inversion
  - use MULGF2 efficiently
  - executed with regularity

HW implementation

- Hardware algorithm for inversion [Brunner et al., ’93]

$S(x) := G(x); R(x) := A(x); \delta := 0;$
for $i = 1$ **to** $2m$ **do**
  if $r_m = 0$ **then**
    $R(x) := x \times R(x);\ \ \delta := \delta + 1;$
  else
    if $s_m = 1$ **then**
      $S(x) := S(x) - R(x);$
    end if
    $S(x) := x \times S(x);$
    if $\delta = 0$ **then**
      $R(x) \leftrightarrow S(x);\ \ \delta := \delta + 1;$
    else
      $\delta := \delta - 1;$
    end if
  end if
end for

1st iteration				$\delta = 0$
S:	x^3	$+\ x^2$		$+\ 1$
R:		x^2		$+\ 1$

HW implementation

- Hardware algorithm for inversion [Brunner et al., '93]

$S(x) := G(x); R(x) := A(x); \delta := 0;$
for $i = 1$ **to** $2m$ **do**
  $\boxed{\textbf{if } r_m = 0 \textbf{ then} \quad R(x) := x \times R(x);\ \delta := \delta + 1;}$
  else
    if $s_m = 1$ **then**
      $S(x) := S(x) - R(x);$
    end if
    $S(x) := x \times S(x);$
    if $\delta = 0$ **then**
      $R(x) \leftrightarrow S(x);\ \delta := \delta + 1;$
    else
      $\delta := \delta - 1;$
    end if
  end if
end for

1st iteration			$\delta = 0$
S:	x^3	$+ \; x^2$	$+ \; 1$
R:		x^2	$+ \; 1$

$R(x) := x \times R(x);$

$\delta := \delta + 1;$

HW implementation

- Hardware algorithm for inversion [Brunner et al., '93]

$S(x) := G(x); R(x) := A(x); \delta := 0;$
for $i = 1$ **to** $2m$ **do**
  if $r_m = 0$ **then**
    $R(x) := x \times R(x);\ \delta := \delta + 1;$
  else
    if $s_m = 1$ **then**
      $S(x) := S(x) - R(x);$
    end if
    $S(x) := x \times S(x);$
    if $\delta = 0$ **then**
      $R(x) \leftrightarrow S(x);\ \delta := \delta + 1;$
    else
      $\delta := \delta - 1;$
    end if
  end if
end for

1st iteration $\delta = 0$

S: $x^3 + x^2 + 1$
R: $x^2 + 1$

2nd iteration $\delta = 1$

S: $x^3 + x^2 + 1$
R: $x^3 + x$

HW implementation

- Hardware algorithm for inversion [Brunner et al., '93]

$S(x) := G(x); R(x) := A(x); \delta := 0;$
for $i = 1$ **to** $2m$ **do**
  if $r_m = 0$ **then**
    $R(x) := x \times R(x);\ \delta := \delta + 1;$
  else
    if $s_m = 1$ **then**
      $S(x) := S(x) - R(x);$
    end if
    $S(x) := x \times S(x);$
    if $\delta = 0$ **then**
      $R(x) \leftrightarrow S(x);\ \delta := \delta + 1;$
    else
      $\delta := \delta - 1;$
    end if
  end if
end for

1st iteration $\delta = 0$
S: $x^3 + x^2 + 1$
R: $x^2 + 1$

2nd iteration $\delta = 1$
S: $x^3 + x^2 + 1$
R: $x^3 + x$

$S(x) := x \times (S(x) - R(x));$
$\delta := \delta - 1;$

HW implementation

- Hardware algorithm for inversion [Brunner et al., ’93]

$S(x) := G(x); R(x) := A(x); \delta := 0;$
for $i = 1$ **to** $2m$ **do**
  if $r_m = 0$ **then**
    $R(x) := x \times R(x);\ \ \delta := \delta + 1;$
  else
    if $s_m = 1$ **then**
      $S(x) := S(x) - R(x);$
    end if
    $S(x) := x \times S(x);$
    if $\delta = 0$ **then**
      $R(x) \leftrightarrow S(x);\ \ \delta := \delta + 1;$
    else
      $\delta := \delta - 1;$
    end if
  end if
end for

1st iteration $\delta = 0$

S:	x^3	+	x^2			+	1
R:			x^2			+	1

2nd iteration $\delta = 1$

S:	x^3	+	x^2			+	1
R:	x^3			+	x		

3rd iteration $\delta = 0$

S:	x^3	+	x^2	+	x		
R:	x^3			+	x		

HW implementation

- Hardware algorithm for inversion [Brunner et al., '93]

$S(x) := G(x); R(x) := A(x); \delta := 0;$
for $i = 1$ **to** $2m$ **do**
  if $r_m = 0$ **then**
    $R(x) := x \times R(x);\ \ \delta := \delta + 1;$
  else
    if $s_m = 1$ **then**
      $S(x) := S(x) - R(x);$
    end if
    $S(x) := x \times S(x);$
    if $\delta = 0$ **then**
      $R(x) \leftrightarrow S(x);\ \ \delta := \delta + 1;$
    else
      $\delta := \delta - 1;$
    end if
  end if
end for

1st iteration $\delta = 0$
S: $x^3 + x^2 + 1$
R: $x^2 + 1$

2nd iteration $\delta = 1$
S: $x^3 + x^2 + 1$
R: $x^3 + x$

3rd iteration $\delta = 0$
S: $x^3 + x^2 + x$
R: $x^3 + x$

$S(x) := x \times (S(x) - R(x));$
$S(x) <-> R(x);$
$\delta := \delta + 1;$

HW implementation

- Hardware algorithm for inversion [Brunner et al., ’93]

$S(x) := G(x); R(x) := A(x); \delta := 0;$
for $i = 1$ **to** $2m$ **do**
  if $r_m = 0$ **then**
    $R(x) := x \times R(x);\ \delta := \delta + 1;$
  else
    if $s_m = 1$ **then**
      $S(x) := S(x) - R(x);$
    end if
    $S(x) := x \times S(x);$
    if $\delta = 0$ **then**
      $R(x) \leftrightarrow S(x);\ \delta := \delta + 1;$
    else
      $\delta := \delta - 1;$
    end if
  end if
end for

1st iteration $\delta = 0$
S: $x^3 + x^2 + 1$
R: $x^2 + 1$

2nd iteration $\delta = 1$
S: $x^3 + x^2 + 1$
R: $x^3 + x$

3rd iteration $\delta = 0$
S: $x^3 + x^2 + x$
R: $x^3 + x$

4th iteration $\delta = 1$
S: $x^3 + x$
R: x^3

Main Idea 2

- Operations corresponding to contiguous k iterations of Brunner's algorithm can be represented as

$$\begin{pmatrix} R(x) & U(x) \\ S(x) & V(x) \end{pmatrix} := H(x) \times \begin{pmatrix} R(x) & U(x) \\ S(x) & V(x) \end{pmatrix};$$

  - Each element of the matrix $H(x)$ is a polynomial with degree less than or equal to k on $\mathrm{GF}(2)$

The Matrix $H(x)$ (1/2)

1st iteration $\delta = 0$

S:	x^3	$+$	x^2	$+$	1
R:			x^2	$+$	1

$R(x) := x \times R(x);$

$\delta := \delta + 1;$

- The operation is represented in matrices as

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := \begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix};$$

The Matrix $H(x)$ (1/2)

1st iteration $\delta = 0$

S: $x^3 + x^2 + 1$

R: $x^2 + 1$

2nd iteration $\delta = 1$

S: $x^3 + x^2 + 1$

R: $x^3 + x$

$S(x) := x \times (S(x) - R(x));$

$\delta := \delta - 1;$

- The operations are represented in matrices as

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := \begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix};$$

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := \begin{pmatrix} 1 & 0 \\ x & x \end{pmatrix} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix};$$

The Matrix $H(x)$ (1/2)

1st iteration $\delta = 0$

$S:\quad x^3 + x^2 + 1$

$R:\quad x^2 + 1$

2nd iteration $\delta = 1$

$S:\quad x^3 + x^2 + 1$

$R:\quad x^3 + x$

3rd iteration $\delta = 0$

$S:\quad x^3 + x^2 + x$

$R:\quad x^3 + x$

$S(x) := x \times (S(x) - R(x));$

$S(x) <-> R(x);$

$\delta := \delta + 1;$

- The operations are represented in matrices as

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := \begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix};$$

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := \begin{pmatrix} 1 & 0 \\ x & x \end{pmatrix} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix};$$

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := \begin{pmatrix} x & x \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix};$$

The Matrix $H(x)$ (2/2)

- The operations in these three iterations can be represented as

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := \boxed{\begin{pmatrix} x & x \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ x & x \end{pmatrix} \times \begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix}} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix};$$

$$= \begin{pmatrix} x^3 + x^2 & x^2 \\ x & 0 \end{pmatrix} = H(x)$$

- By using $H(x)$
  - We can calculate the operations in these three iterations at once
  - We can use MULGF2 instruction efficiently

New Algorithm

1. calculates $H(x)$ from the most significant word of $R(x)$ and $S(x)$
   - with only single-word operations
2. calculates

$$\begin{pmatrix} R(x) & U(x) \\ S(x) & V(x) \end{pmatrix} := H(x) \times \begin{pmatrix} R(x) & U(x) \\ S(x) & V(x) \end{pmatrix};$$

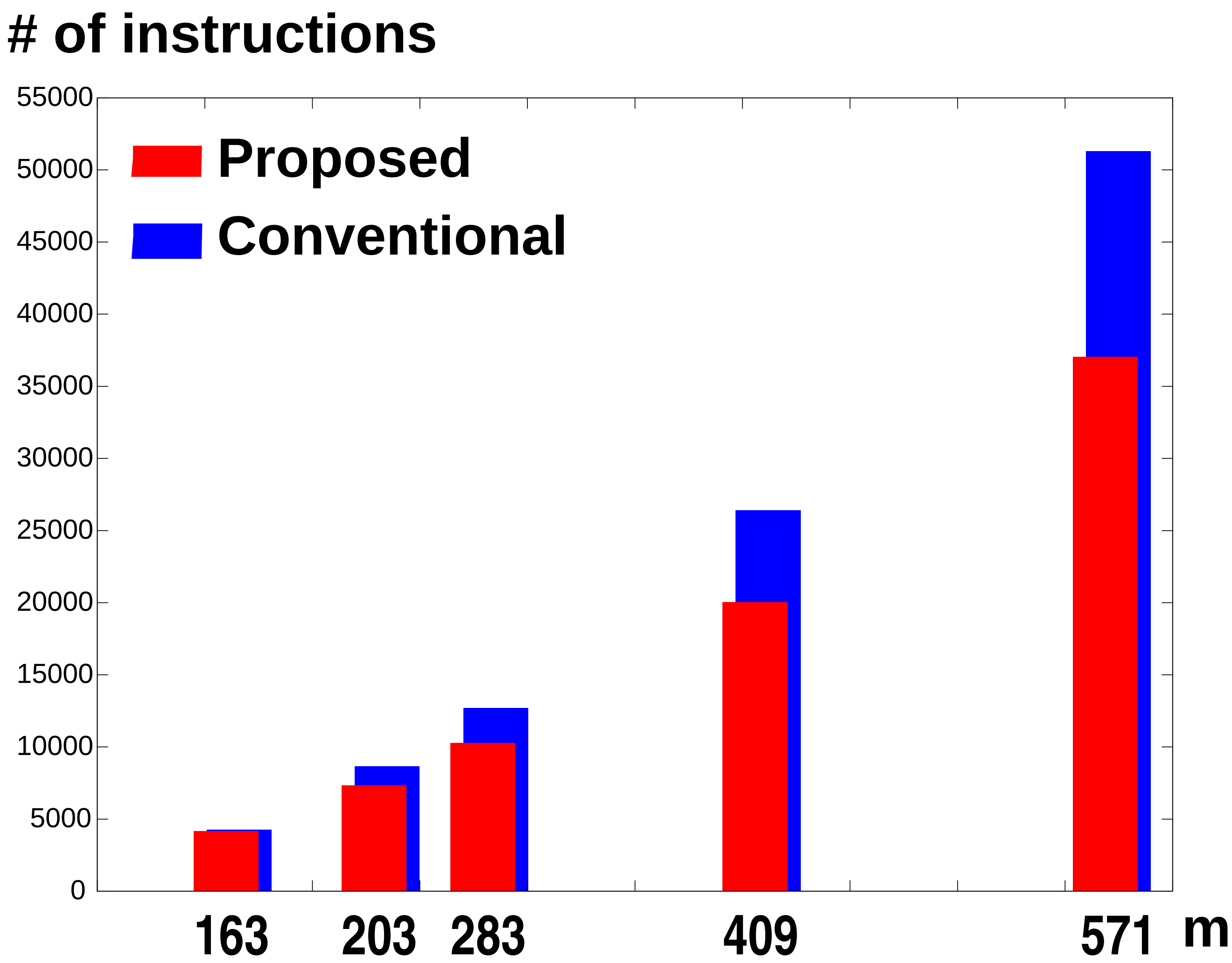
   efficiently by using MULGF2
3. continues the process until $R(x)$ becomes 0

Evaluation

- We compared # of MULGF2 and XOR instructions of the proposed algorithm with that of the conventional one
- Assumption
  - We compared average # of instructions for executing inversion of $1,000$ random elements
  - We counted instructions for multi-word operations in two algorithms
  - MULGF2 has single cycle latency

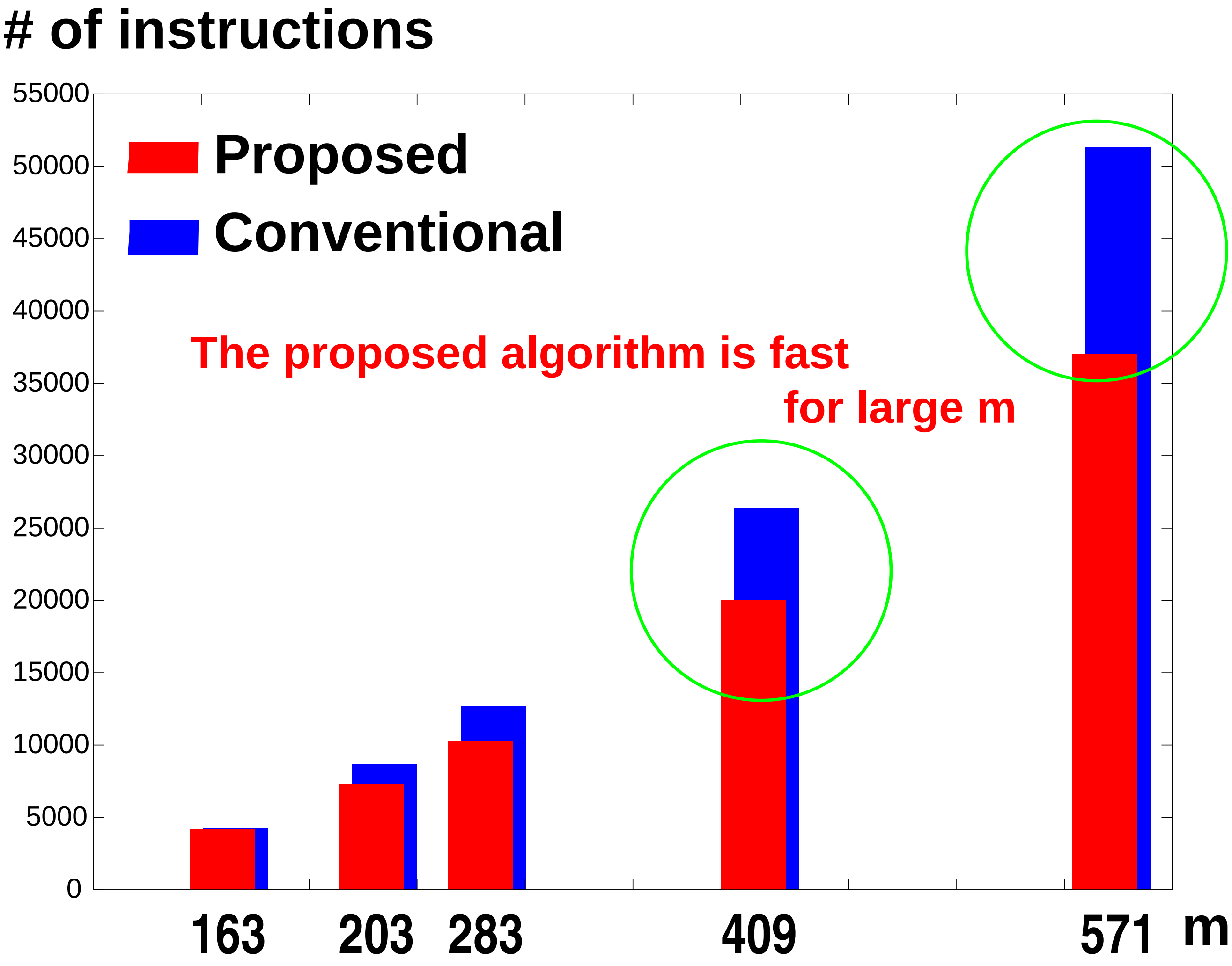
Comparison of # of instruction (1/2)

- the word size of a processor $= 16$



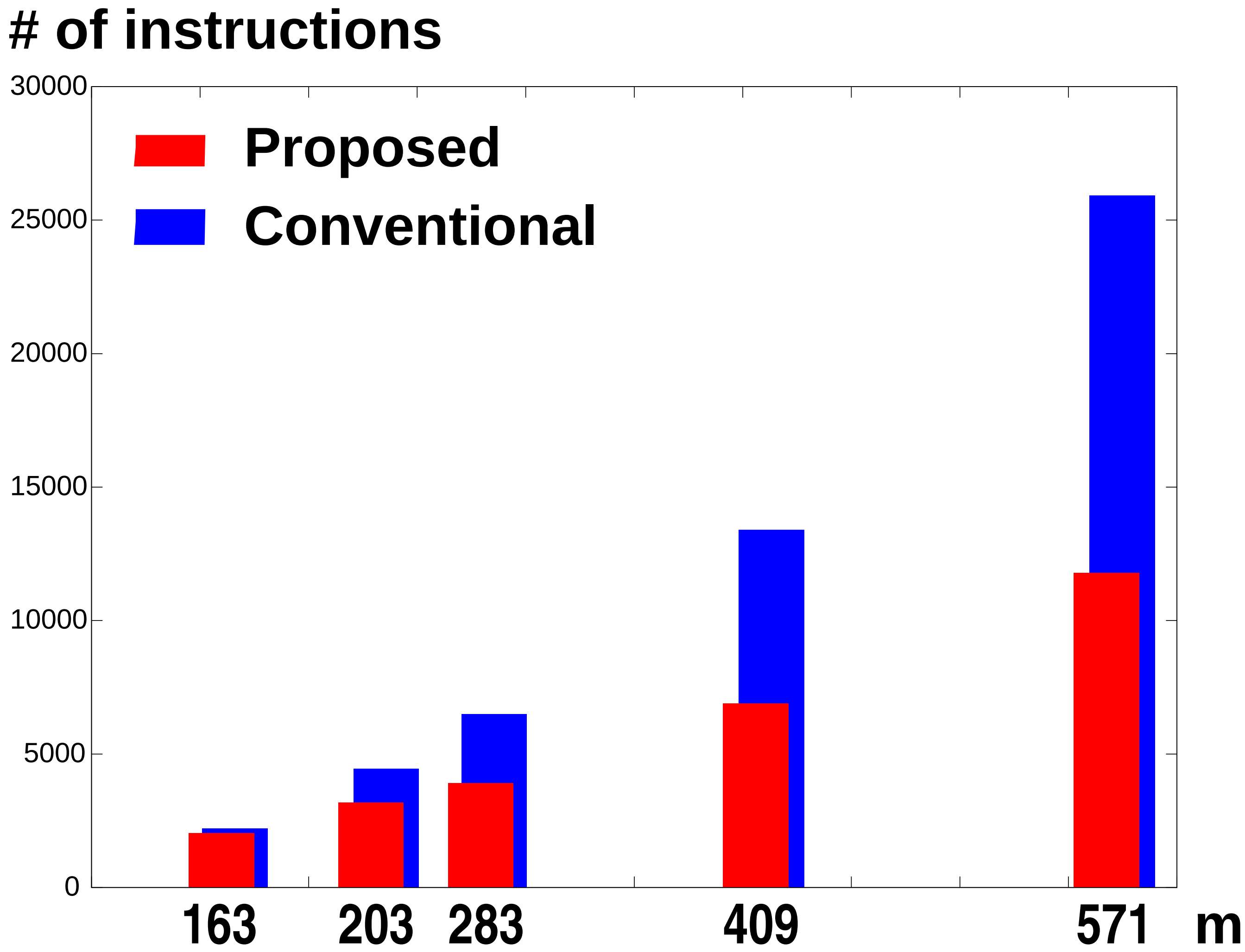
Comparison of # of instruction (1/2)

- the word size of a processor $= 16$



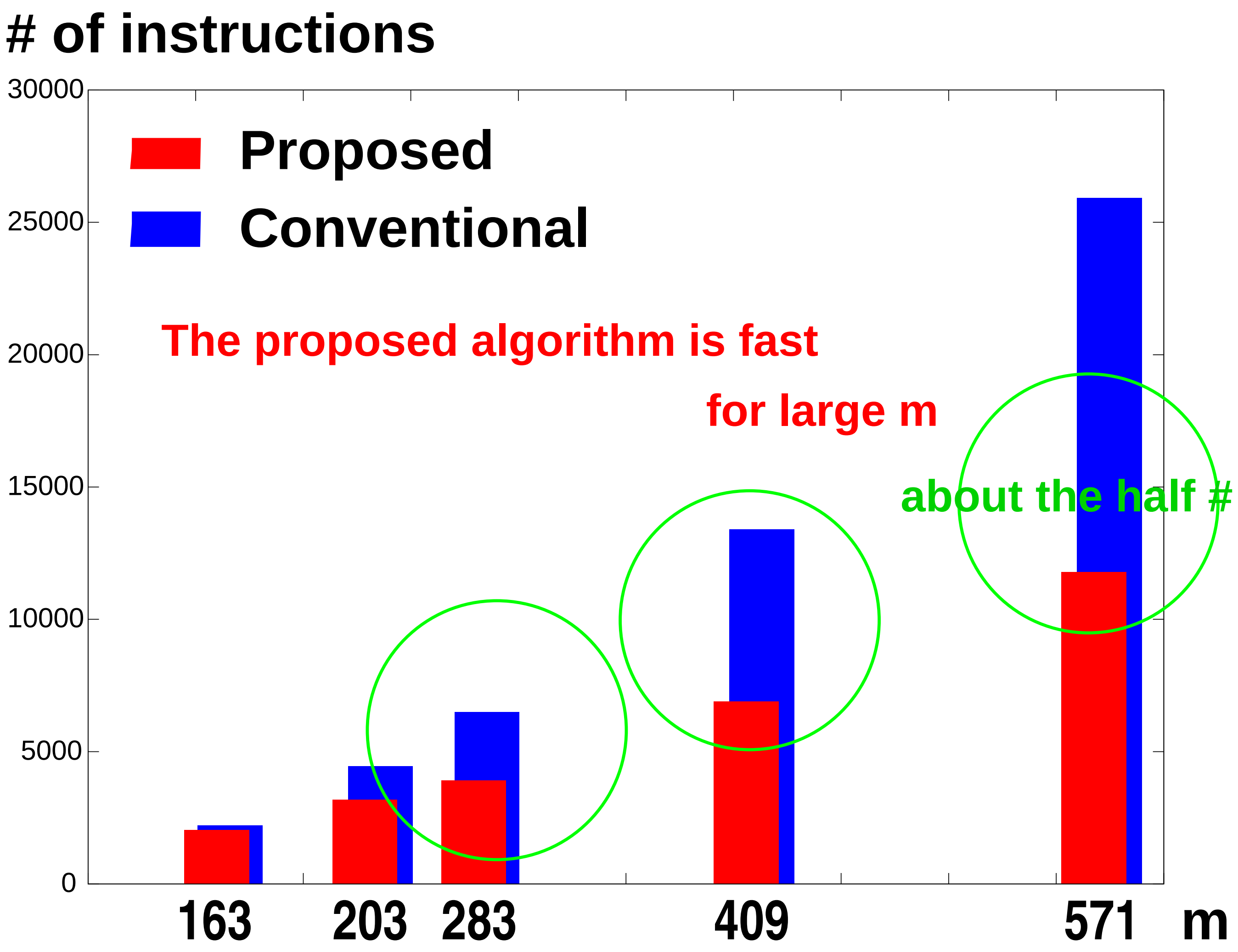
Comparison of # of instruction (2/2)

- the word size of a processor $= 32$



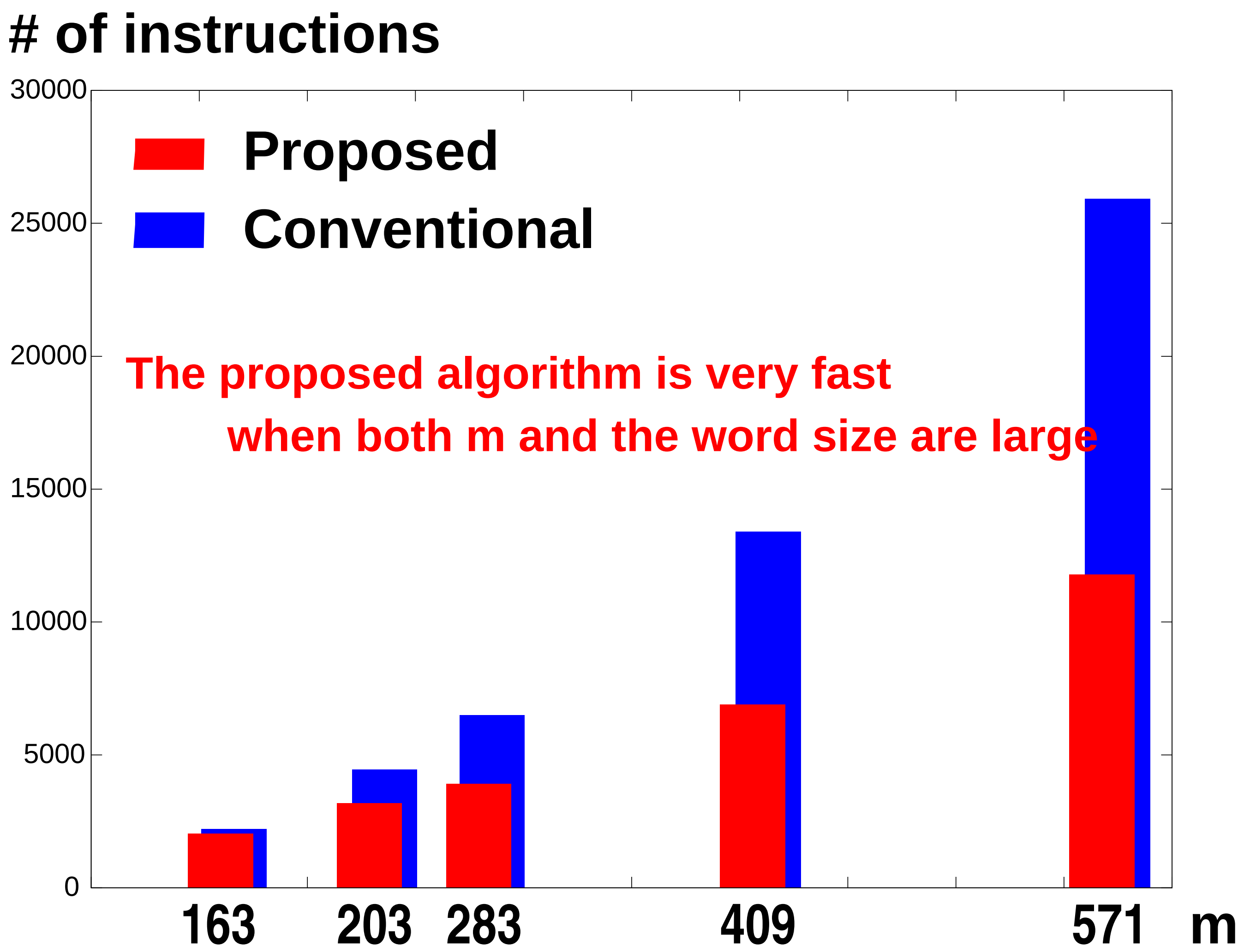
Comparison of # of instruction (2/2)

- the word size of a processor $= 32$



Comparison of # of instruction (2/2)

- the word size of a processor $= 32$



Concluding Remarks

- We have proposed a new algorithm for inversion in $\mathrm{GF}(2^m)$
  - the matrix $H(x)$
    - represents operations corresponding to several contiguous iterations of Brunner's algorithm
    - obtained with only single-word operation
  - suitable for implementation using MULGF2
  - executed with regularity
- When both m and the word size of a processor are large
  - the proposed algorithm can execute inversion very fast

Thank you for listening!