PhiRSA: Exploiting the Computing Power of Vector Instructions on Intel Xeon Phi for RSA

***Yuan Zhao*[1]**, Wuqiong Pan[1], Jingqiang Lin[1], Peng Liu[2], Cong Xue[1], Fangyu Zheng[1]

[1]*Institute of Information Engineering, Chinese Academy of Sciences*
[2]*College of Information Sciences and Technology, The Pennsylvania State University*

Outline

Background

中國科學院 信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

PennState
College of Information
Sciences and Technology

DCS Center
DATA ASSURANCE & COMMUNICATION SECURITY
EST 1980

Cryptography Engineering

- Software implementation of cryptographic algorithm
- Performance
  - Throughput and latency
  - Fully exploiting the features of processors

Computing Power of Processors

- CPU
  - single-instruction-multiple-data (SIMD)

    Intel MMX/SSE/AVX, ARM NEON and AMD 3DNow
  - simultaneous-multithreading (SMT)

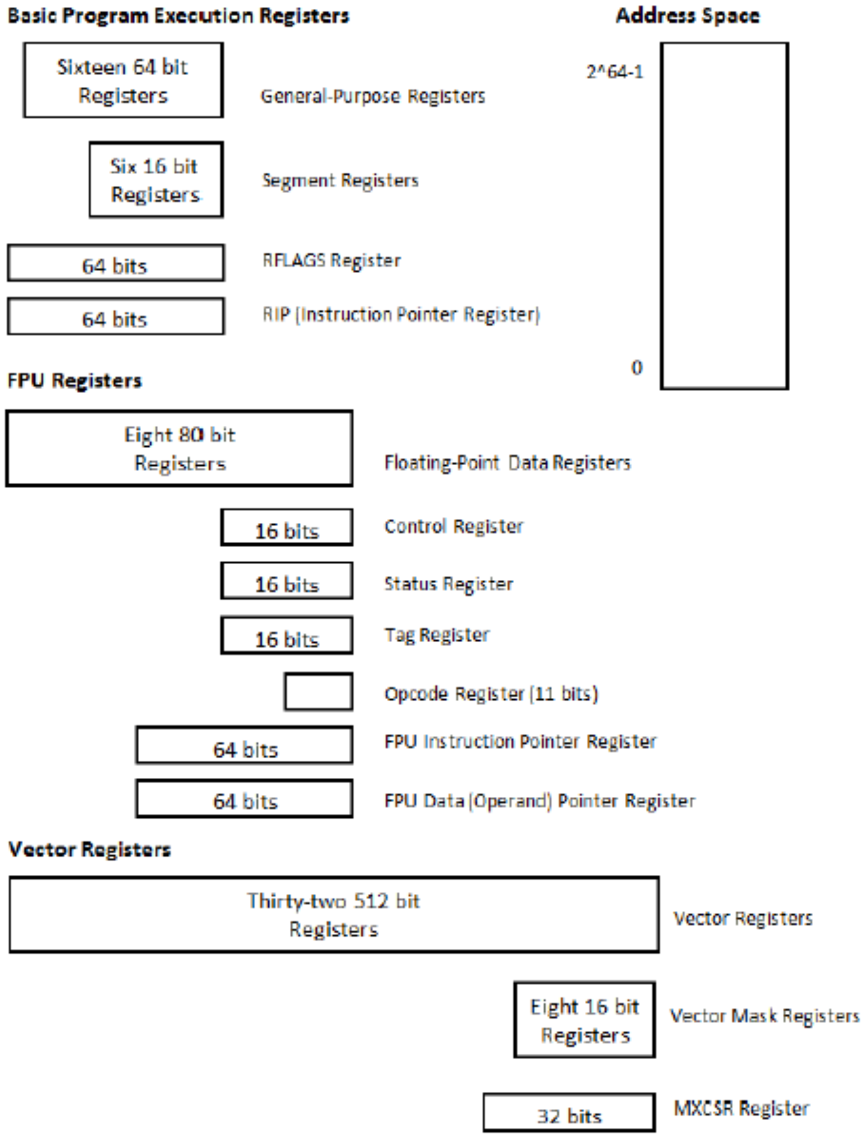
    Intel Hyper-Threading
- GPU
  - single-instruction-multiple-thread (SIMT)

Intel Xeon Phi

- Computing Power
  - 61 cores
  - 512-bit VPU
  - 4 hyperthreads
- Coprocessor OS
- Execution mode
  - Offload execution mode
  - Native execution mode

Intel Xeon Phi

- Registers
  - 16 64-bit scalar registers
  - 32 512-bit vector registers
  - 8 16-bit mask registers



Intel Xeon Phi

- Instruction Set Architecture
  - Mask register: write-mask, carry holder
  - vpmulhud, vpmulld, vpermd, valignd
  - vector-add-with-carry instruction vpadcd

$$vpadcd \quad (zmm2/memory),\ k2,\ zmm1\{k1\}$$

Montgomery Multiplication

Algorithm 1 Montgomery Multiplication CIOS Method[19]

Input: Modulus M, $R = 2^{nw}$, $R > M$, $\gcd(M, R) = 1$, 2^w is radix, n is digits number
$0 \leqslant A, B < M$, $B = \sum_{i=0}^{n-1} b_i 2^{iw}$, $\mu = -M^{-1} \bmod 2^w$

Output: $S = A \cdot B \cdot R^{-1} \pmod{M}$, $0 \leqslant S < M$.

```
 1: $S \leftarrow 0$
 2: for i from 0 to n-1 do
 3:     $S \leftarrow S + A \cdot b_i$
 4:     $q \leftarrow S[0] \cdot \mu \bmod 2^w$
 5:     $S \leftarrow S + M \cdot q$
 6:     $S \leftarrow S / 2^w$
 7: end for
 8: if $S \geqslant M$ then
 9:     $S \leftarrow S - M$
10: end if
11: return $S$
```

Related Work

- Storing the large integers in vectors horizontally for fine-grained parallel

  Redundant representation

- Splitting the Montgomery multiplication into two parts to compute in parallel
- Computing multiple Montgomery multiplications simultaneously in vector elements

Montgomery Multiplication Design

中国科学院 信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

PennState
College of Information Sciences and Technology

DCS CENTER
DATA ASSURANCE & COMMUNICATION SECURITY
EST 1980

Vector Carry Propagation Chain (VCPC)

- A group of vectors S, L and a carry k1 are added together in a chain to propagate k1 forward in a element after each round

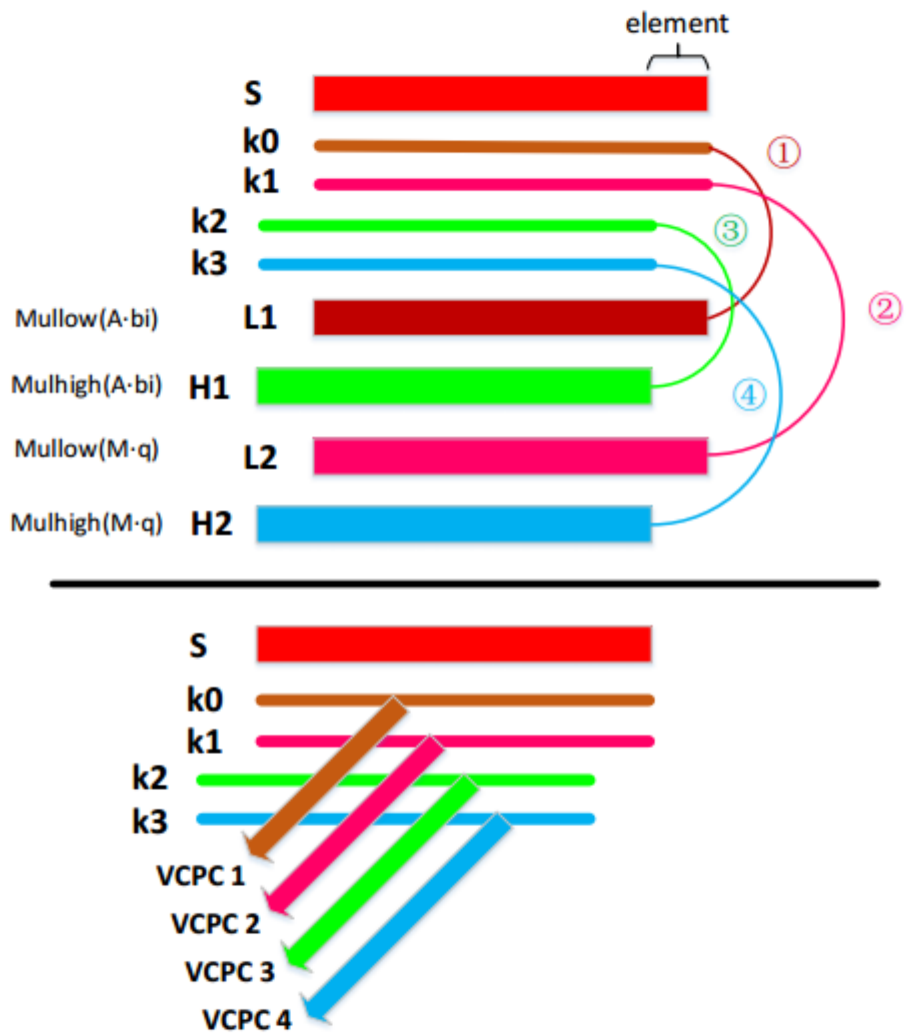
```
$for\ i\ from\ 0\ to\ n-1$
    $T \leftarrow Broadcast(b_i)$
    $L \leftarrow Mullow(A,\ T)$
    $S \leftarrow Vadc(S,\ k1,\ L)$
    $S \leftarrow Rshift(Zero,\ S,\ 1)$
```

Four VCPCs

```
for i from 0 to n-1 do
    $S \leftarrow S + A \cdot b_i$
    $q \leftarrow S[0] \cdot \mu \bmod 2^w$
    $S \leftarrow S + M \cdot q$
    $S \leftarrow S / 2^w$
```

```
$for\ i\ from\ 0\ to\ n-1$
    $L \leftarrow Mullow(A, b_i)$
    $S \leftarrow Vadc(S, k0, L)$
    $L \leftarrow Mullow(M, q)$
    $S \leftarrow Vadc(S, k1, L)$
    $S \leftarrow Rshift(Zero, S, 1)$
    $H \leftarrow Mulhigh(A, b_i)$
    $S \leftarrow Vadc(S, k2, H)$
    $H \leftarrow Mulhigh(M, q)$
    $S \leftarrow Vadc(S, k3, H)$
```

Four VCPCs


element
S
k0
k1
k2
k3
Mullow(A·bi)
L1
Mulhigh(A·bi)
H1
Mullow(M·q)
L2
Mulhigh(M·q)
H2
①
②
③
④
S
k0
k1
k2
k3
VCPC 1
VCPC 2
VCPC 3
VCPC 4

Handling Tail

```
$(T, k0) \leftarrow Vadc(Zero, k0, Zero)$
$(S, k1) \leftarrow Vadc(S, k1, T)$
$(T, k1) \leftarrow Vadc(Zero, k1, Zero)$
$(T, k2) \leftarrow Vadc(T, k2, Zero)$
$H \leftarrow Rshift(Zero, S, 1)$
$(H, k3) \leftarrow Vadc(H, k3, T)$
for i from 0 to $n-2$ do
    if $k3 = 0$ then
        BREAK
    end if
    $(H, k3) \leftarrow Vadc(H, k3, Zero)$
    $k3 \leftarrow Lmove(k3)$
end for
$S \leftarrow Rshift(S, zero, 1)$
$S \leftarrow Rshift(H, S, n-1)$
```

Computing q

- We use carry k1 as write-mask
- Our method does not require an extra move instruction and an extra vector mask register

$$Q \leftarrow Mullow(S,\ U)$$

$$Q \leftarrow Vadd(Q,\ U)\{k1\}$$

Performance Analysis

- Compared with redundant representation

	RR Method	VCPC method
Vector Number	$2 * \lceil l/(s - t * n) \rceil$	$\lceil l/s \rceil$
Instructions/Round	$10 * \lceil l/(s - t * n) \rceil + 4$	$9 * \lceil l/s \rceil + 4$
Round	$\lceil l/(w - t) \rceil$	$\lceil l/w \rceil$
Instructions	$\lceil l/(w - t) \rceil * (10 * \lceil l/(s - t * n) \rceil + 4)$	$\lceil l/w \rceil * (9 * \lceil l/s \rceil + 4)$

- For 1024-bit Montgomery multiplication on Intel Xeon Phi, VCPC method requires 704 instructions, while RR method 1224 instructions. VCPC method only needs a factor of 0.58 instructions than RR method

PhiRSA Implementation

中國科學院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

PennState
College of Information
Sciences and Technology

DCS CENTER
DATA ASSURANCE & COMMUNICATION SECURITY
EST 1980

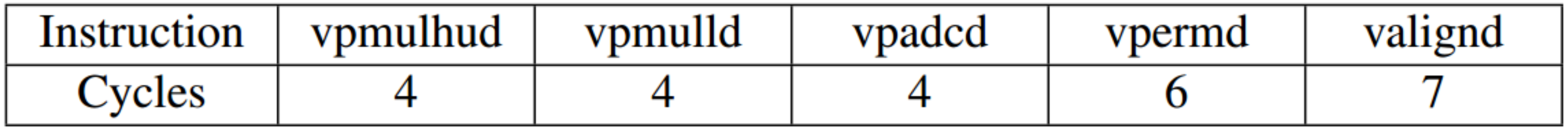
Montgomery Multiplication

- Implementation Issues
  - Making VPUs fully pipelined
  - Maintaining Carry Bits in Vector Mask Registers

Latencies of Vector Instructions

- Four-cycle instructions (vpmulhud, vpmulld and vpadcd) can be fully pipelined by four hyperthreads
- If vpermd and valignd do not use the data produced by the prior instruction, they can be fully pipelined

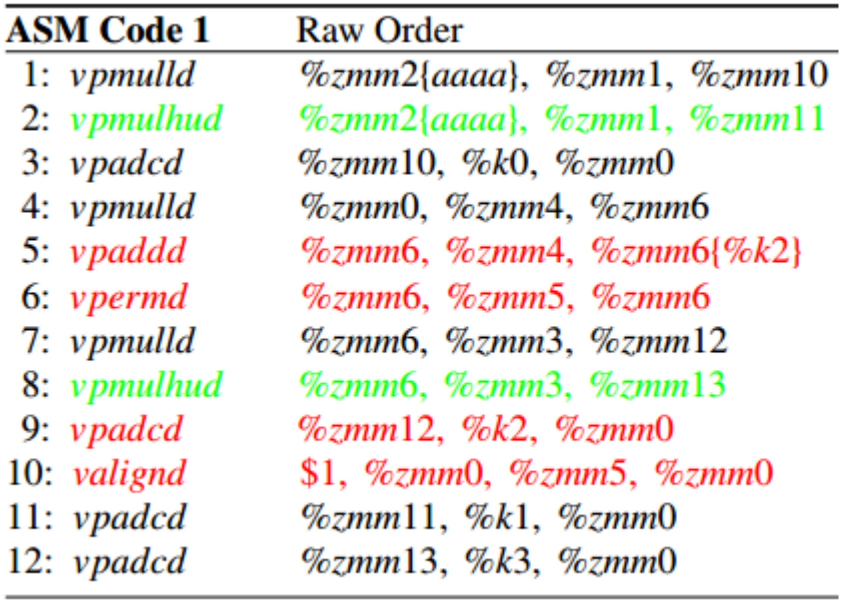
Table 2. The Latencies of Vector Instructions On Intel Xeon Phi



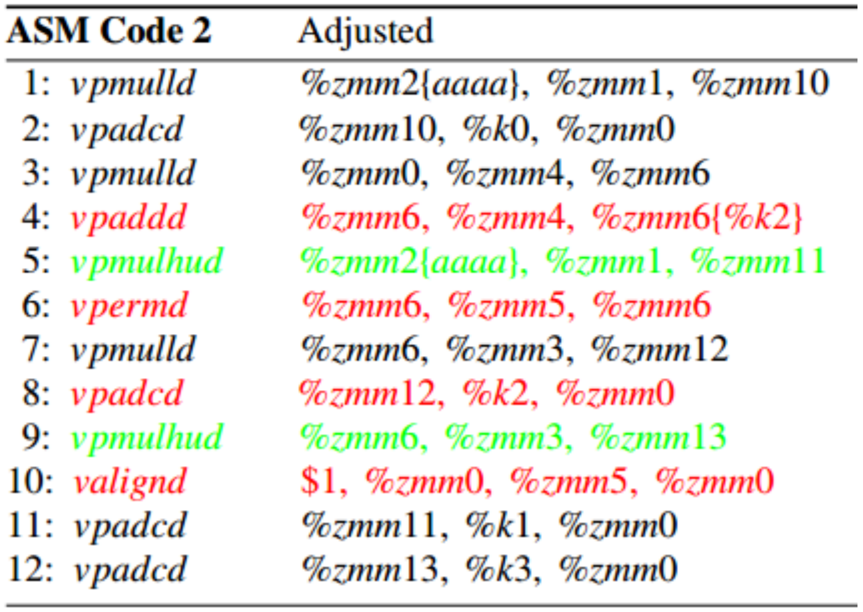
Instruction	vpmulhud	vpmulld	vpadcd	vpermd	valignd
Cycles	4	4	4	6	7

Adjust the Sequence of Instructions

- Raw order code requires 15.6 cycles, the code after adjusting only requires 12.2 cycles which makes the utilization of VPU reach 98%



ASM Code 1	Raw Order
1: *vpmulld*	*%zmm2{aaaa}, %zmm1, %zmm10*
2: *vpmulhud*	*%zmm2{aaaa}, %zmm1, %zmm11*
3: *vpadcd*	*%zmm10, %k0, %zmm0*
4: *vpmulld*	*%zmm0, %zmm4, %zmm6*
5: *vpaddd*	*%zmm6, %zmm4, %zmm6{%k2}*
6: *vpermd*	*%zmm6, %zmm5, %zmm6*
7: *vpmulld*	*%zmm6, %zmm3, %zmm12*
8: *vpmulhud*	*%zmm6, %zmm3, %zmm13*
9: *vpadcd*	*%zmm12, %k2, %zmm0*
10: *valignd*	*$1, %zmm0, %zmm5, %zmm0*
11: *vpadcd*	*%zmm11, %k1, %zmm0*
12: *vpadcd*	*%zmm13, %k3, %zmm0*



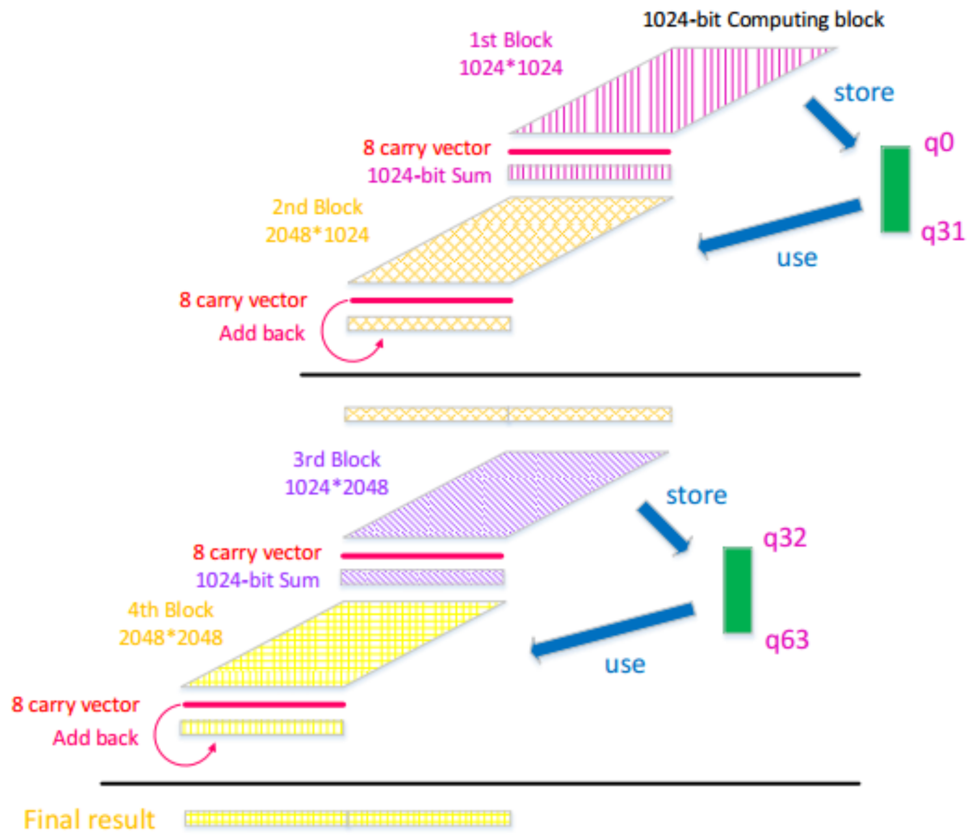
ASM Code 2	Adjusted
1: *vpmulld*	*%zmm2{aaaa}, %zmm1, %zmm10*
2: *vpadcd*	*%zmm10, %k0, %zmm0*
3: *vpmulld*	*%zmm0, %zmm4, %zmm6*
4: *vpaddd*	*%zmm6, %zmm4, %zmm6{%k2}*
5: *vpmulhud*	*%zmm2{aaaa}, %zmm1, %zmm11*
6: *vpermd*	*%zmm6, %zmm5, %zmm6*
7: *vpmulld*	*%zmm6, %zmm3, %zmm12*
8: *vpadcd*	*%zmm12, %k2, %zmm0*
9: *vpmulhud*	*%zmm6, %zmm3, %zmm13*
10: *valignd*	*$1, %zmm0, %zmm5, %zmm0*
11: *vpadcd*	*%zmm11, %k1, %zmm0*
12: *vpadcd*	*%zmm13, %k3, %zmm0*

2048-bit Montgomery Multiplication

- 2048-bit Montgomery multiplication has sixteen VCPCs, it will produce sixteen carry vector every round
- Intel Xeon Phi only has eight vector mask registers
- Using instruction kmov to move carries between vector mask registers and general purpose registers will rouse gigantic performance loss

2048-bit Montgomery Multiplication

- We split 2048-bit Montgomery multiplication into four parts which are similar to 1024-bit Montgomery multiplication implementation



Evaluation

Evaluation

- Platform

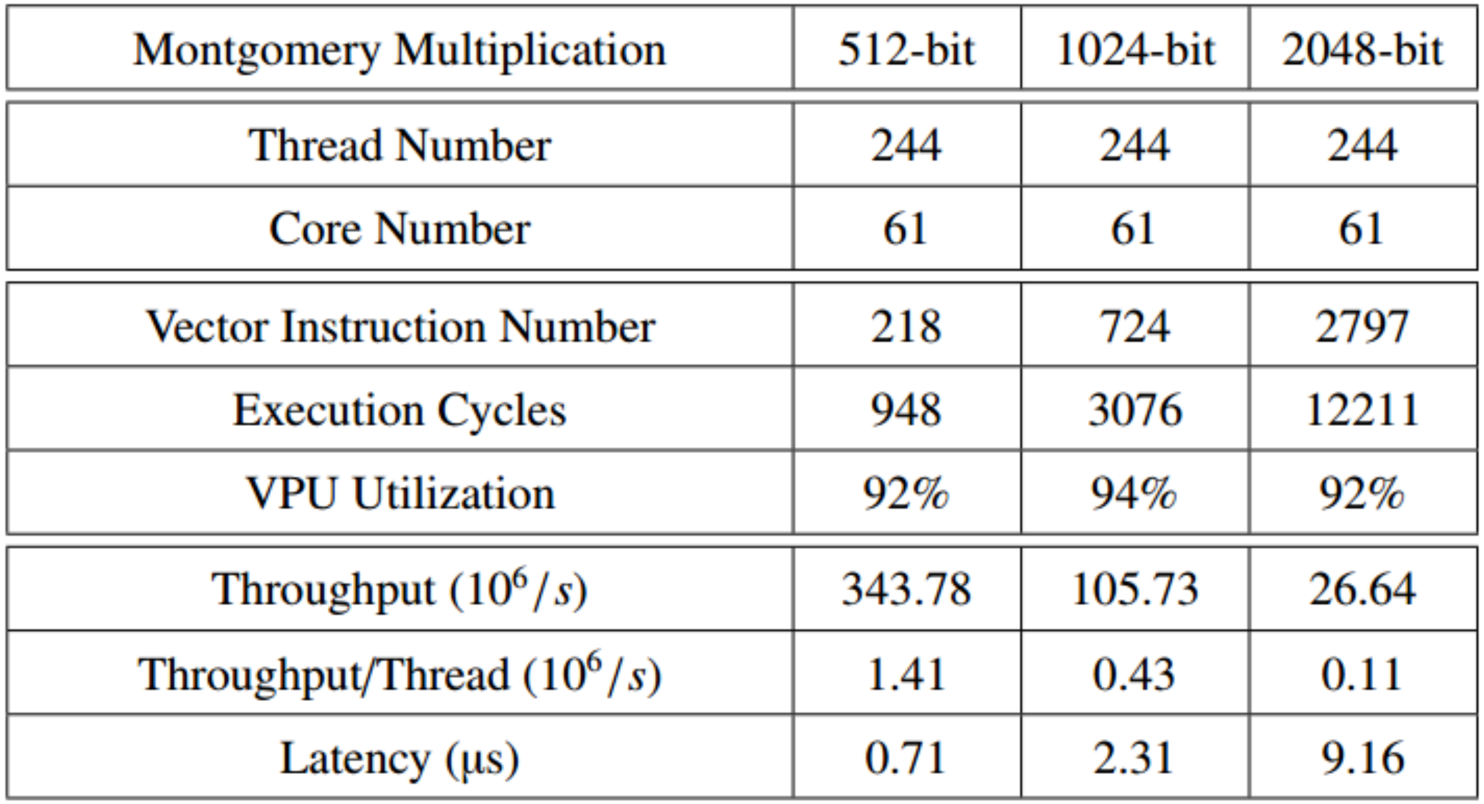
  Coprocessor: Intel Xeon Phi 7120P

  Host: Intel Xeon E5 2697v2, RedHat 6.4, Intel Composer XE 2013.

- Performance
  - Implementation results
  - Comparisons with the previous works on Intel Xeon Phi
  - Comparisons with the implementations on CPUs and GPUs

Implementation Results

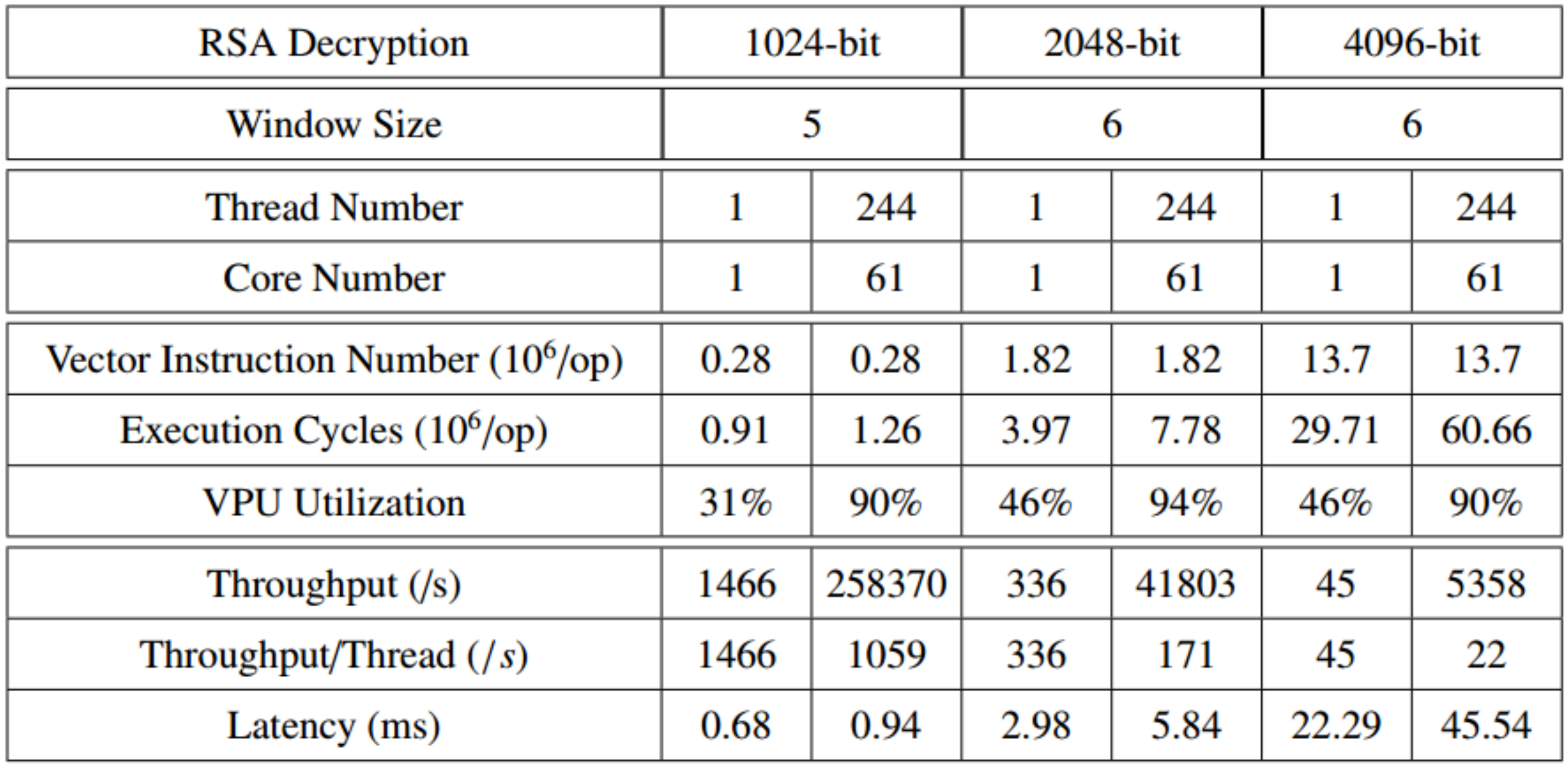
- Montgomery Multiplication



Montgomery Multiplication	512-bit	1024-bit	2048-bit
Thread Number	244	244	244
Core Number	61	61	61
Vector Instruction Number	218	724	2797
Execution Cycles	948	3076	12211
VPU Utilization	92%	94%	92%
Throughput ($10^6/s$)	343.78	105.73	26.64
Throughput/Thread ($10^6/s$)	1.41	0.43	0.11
Latency (μs)	0.71	2.31	9.16

Implementation Results

- RSA Decryption



RSA Decryption	1024-bit		2048-bit		4096-bit	
Window Size	5		6		6	
Thread Number	1	244	1	244	1	244
Core Number	1	61	1	61	1	61
Vector Instruction Number (10^6/op)	0.28	0.28	1.82	1.82	13.7	13.7
Execution Cycles (10^6/op)	0.91	1.26	3.97	7.78	29.71	60.66
VPU Utilization	31%	90%	46%	94%	46%	90%
Throughput (/s)	1466	258370	336	41803	45	5358
Throughput/Thread ($/s$)	1466	1059	336	171	45	22
Latency (ms)	0.68	0.94	2.98	5.84	22.29	45.54

Previous Works on Intel Xeon Phi

- Comparison with the Implementation of Redundant Representation

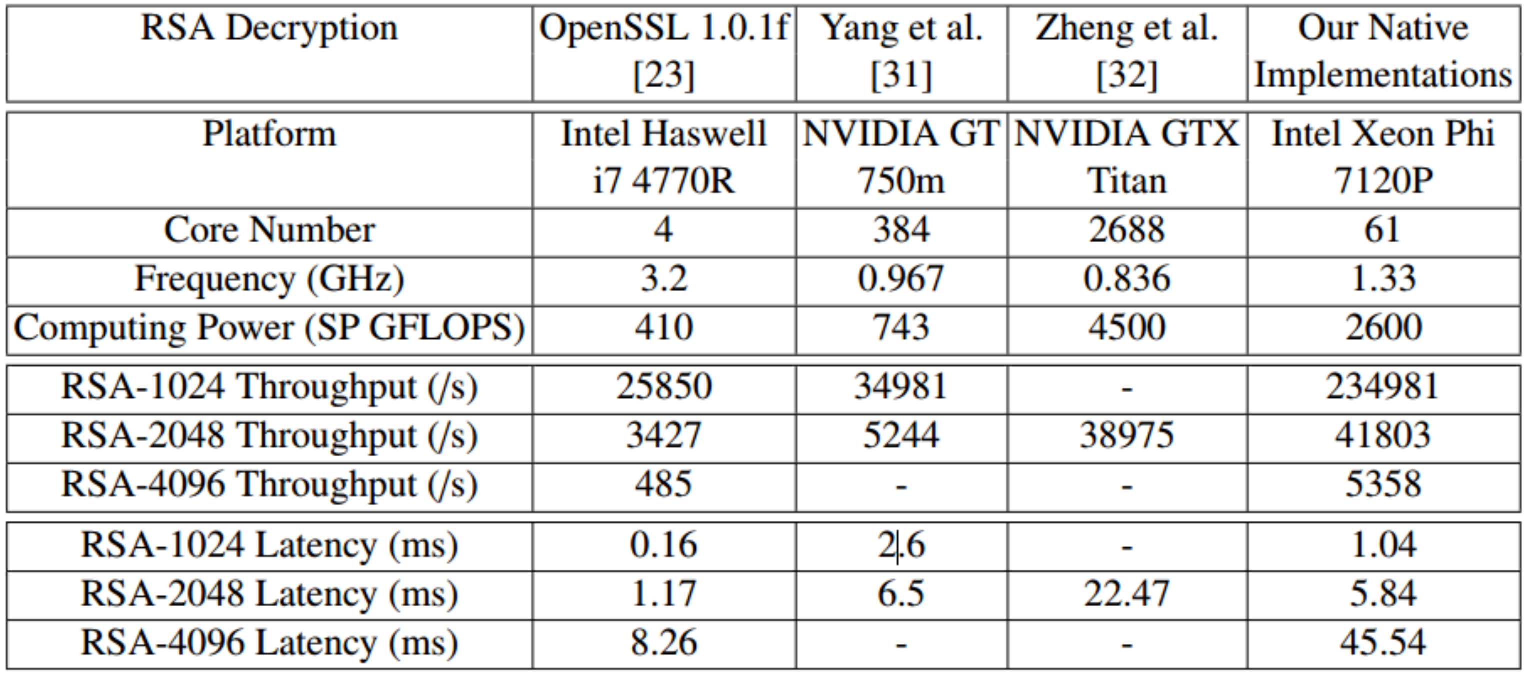
	512-bit MontMul (instructions)	1024-bit MontMul (instructions)	2048-bit MontMul (instructions)
Keliris et al. [12] (Scaled)	3846	9498	28776
Our VCPC Method	218	724	2797

- Comparison with the Implementation of Carry Propagation

	512-bit RSA-1024 Throughput (/s)	1024-bit RSA-2048 Throughput (/s)	2048-bit RSA-4096 Throughput (/s)
Chang et al. [7] (Scaled)	1310	7217	30282
Our VCPC Method	5358	41803	258370

Implementations on CPUs and GPUs

- Compared with OpenSSL on CPUs, the throughput is about 7 times, the latency is no more than 90%
- Compared with the best implementation on GPUs, the throughput is about 1.07 times, the latency is only 26%



RSA Decryption	OpenSSL 1.0.1f [23]	Yang et al. [31]	Zheng et al. [32]	Our Native Implementations
Platform	Intel Haswell i7 4770R	NVIDIA GT 750m	NVIDIA GTX Titan	Intel Xeon Phi 7120P
Core Number	4	384	2688	61
Frequency (GHz)	3.2	0.967	0.836	1.33
Computing Power (SP GFLOPS)	410	743	4500	2600
RSA-1024 Throughput (/s)	25850	34981	-	234981
RSA-2048 Throughput (/s)	3427	5244	38975	41803
RSA-4096 Throughput (/s)	485	-	-	5358
RSA-1024 Latency (ms)	0.16	2.6	-	1.04
RSA-2048 Latency (ms)	1.17	6.5	22.47	5.84
RSA-4096 Latency (ms)	8.26	-	-	45.54

Conclusion

Conclusion

- Our contributions
  - We propose a novel vector-oriented Montgomery multiplication design and implementation to fully exploit the computing power of vector instructions on Intel Xeon Phi, and implement RSA named PhiRSA
  - We demonstrate that Intel Xeon Phi can be used to achieve both high throughput and small latency for RSA

中國科學院 信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

DCS CENTER
DATA ASSURANCE & COMMUNICATION SECURITY
EST 1980

Thank You!

zhaoyuan@iie.ac.cn