

Implementation of public key algorithms in CUDA

Presented by: Hao Wu

Supervisors: Patrick Bours

Maciej Pietka

Outline

- Problem Description
- CUDA
- Methods
- Performance Comparison
- Discussion and Conclusion

Problem Description

- Public key cryptography
 - Modular arithmetic with large numbers
 - Time-consuming
 - Slower than symmetric key algorithms
 - Tradeoff between security and efficiency
- How to make a more efficient and faster implementation of public key algorithms

CUDA-enabled GPU

- General-purpose programming system for NVIDIA CUDA-enabled GPUs (device)
- The massive parallel processing properties
- Compute Unified Device Architecture (CUDA)
- Co-processor to CPU (host)

CUDA

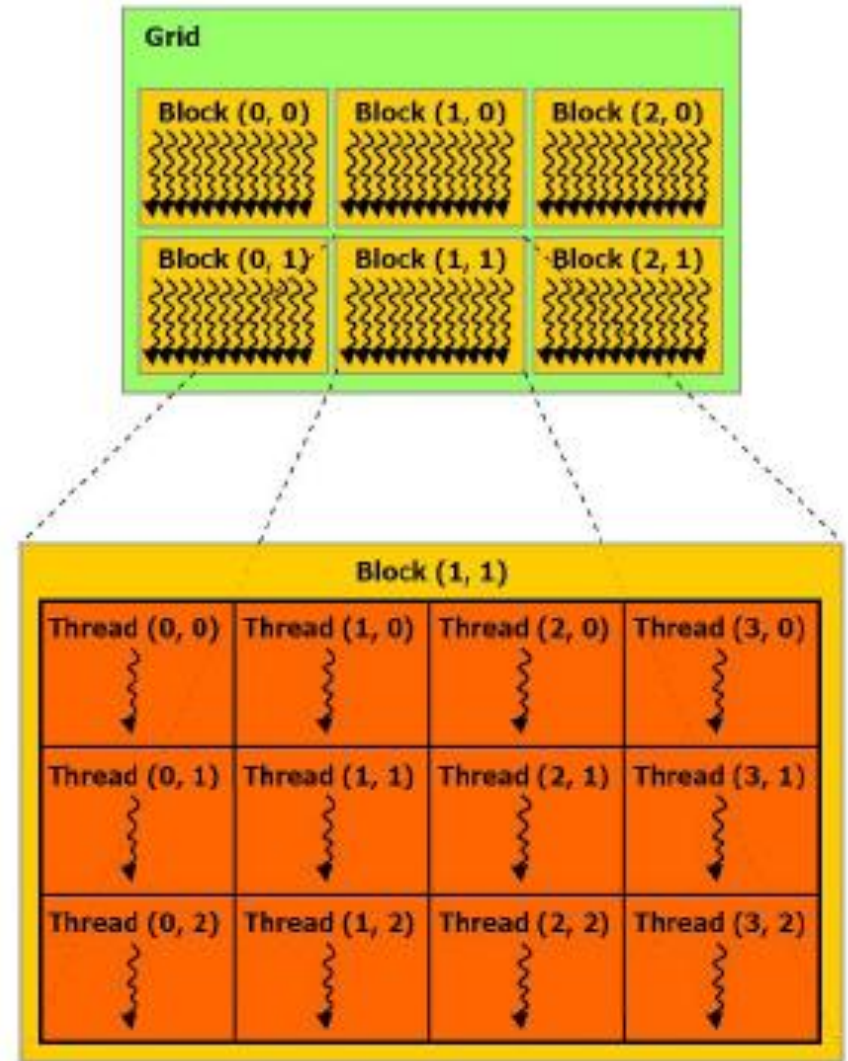
- Kernel in Single Program Multiple Data (SPMD) model
- Own memory and processing elements
- Data transfer between host and device(s)
- Input and Output (IO) time must be considered

- CUDA

- Grid is a group of blocks.
- Block is a group of threads.
- Thread executes a kernel with a given index.

- Problems

- No recursion
- No synchronization between the blocks.
- Debugging using the Device Emulation Mode



Methods

- A representative public-key algorithm --- RSA
- Implemented on CPU and GPU
- Modular arithmetic
- Parallel addition and subtraction
- Parallel right shift
- Parallel multiplication
- Parallel modular exponentiation

Modular arithmetic

- Modular exponentiation and multiplication
- Naïve methods are inefficient
 - Too many subtractions and comparisons
- To reduce numbers modulo a number without performing division
 - Barrett modular reduction
 - Montgomery algorithms

Parallel addition and subtraction

$$X = (X_n \ X_{n-1} \ X_{n-2} \ \cdots \ X_4 \ X_3 \ X_2 \ X_1 \ X_0)_b$$

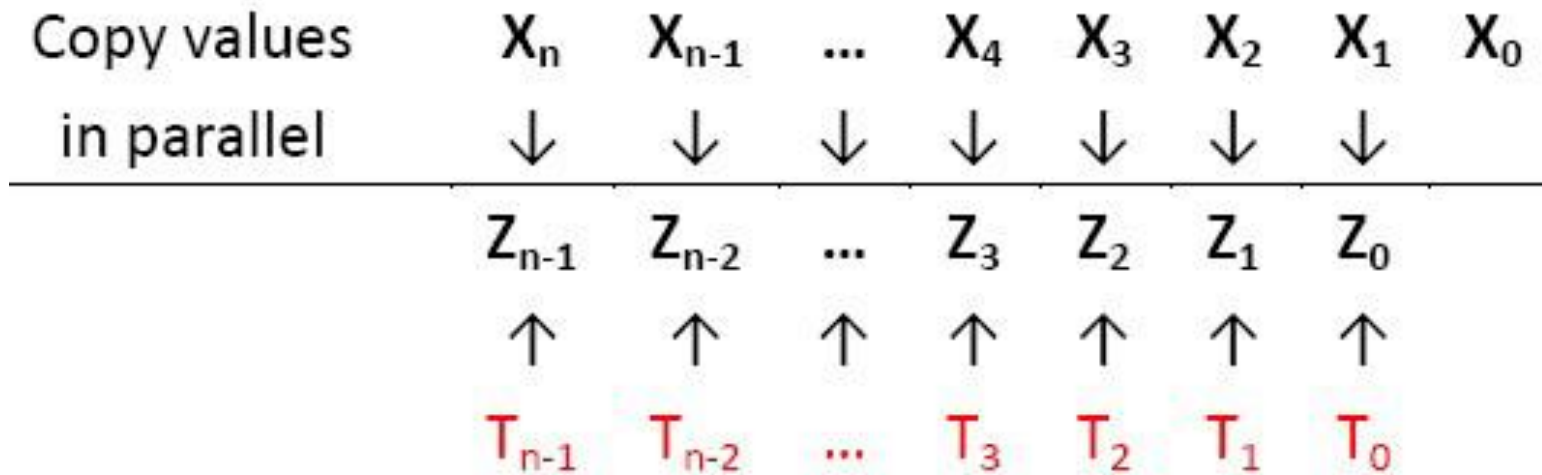
$$Y = (Y_n \ Y_{n-1} \ Y_{n-2} \ \cdots \ Y_4 \ Y_3 \ Y_2 \ Y_1 \ Y_0)_b$$

$$Z_i = X_i \pm Y_i \quad (0 \leq i \leq n) \quad \text{For all threads } T_i \quad (0 \leq i \leq n)$$

	X_n	X_{n-1}	\cdots	X_4	X_3	X_2	X_1	X_0
$+/-$	Y_n	Y_{n-1}	\cdots	Y_4	Y_3	Y_2	Y_1	Y_0
	Z_n	Z_{n-1}	\cdots	Z_4	Z_3	Z_2	Z_1	Z_0
	\uparrow	\uparrow	\uparrow	\uparrow	\uparrow	\uparrow	\uparrow	\uparrow
	T_n	T_{n-1}	\cdots	T_4	T_3	T_2	T_1	T_0

Parallel right shift

- $Z = X / b$
- b is the base of large numbers



Parallel multiplication

- $Z = X Y$
- To compute $X_i Y_j$ in parallel

					X_4	X_3	X_2	X_1	X_0
				\times	Y_4	Y_3	Y_2	Y_1	Y_0
Matrix $R \setminus C$	8	7	6	5	4	3	2	1	0
0					$X_4 Y_0$	$X_3 Y_0$	$X_2 Y_0$	$X_1 Y_0$	$X_0 Y_0$
1				$X_4 Y_1$	$X_3 Y_1$	$X_2 Y_1$	$X_1 Y_1$	$X_0 Y_1$	
2			$X_4 Y_2$	$X_3 Y_2$	$X_2 Y_2$	$X_1 Y_2$	$X_0 Y_2$		
3		$X_4 Y_3$	$X_3 Y_3$	$X_2 Y_3$	$X_1 Y_3$	$X_0 Y_3$			
4	$X_4 Y_4$	$X_3 Y_4$	$X_2 Y_4$	$X_1 Y_4$	$X_0 Y_4$				
	Z_8	Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0

Parallel multiplication

$$X_4 Y_1$$

$$X_3 Y_2$$

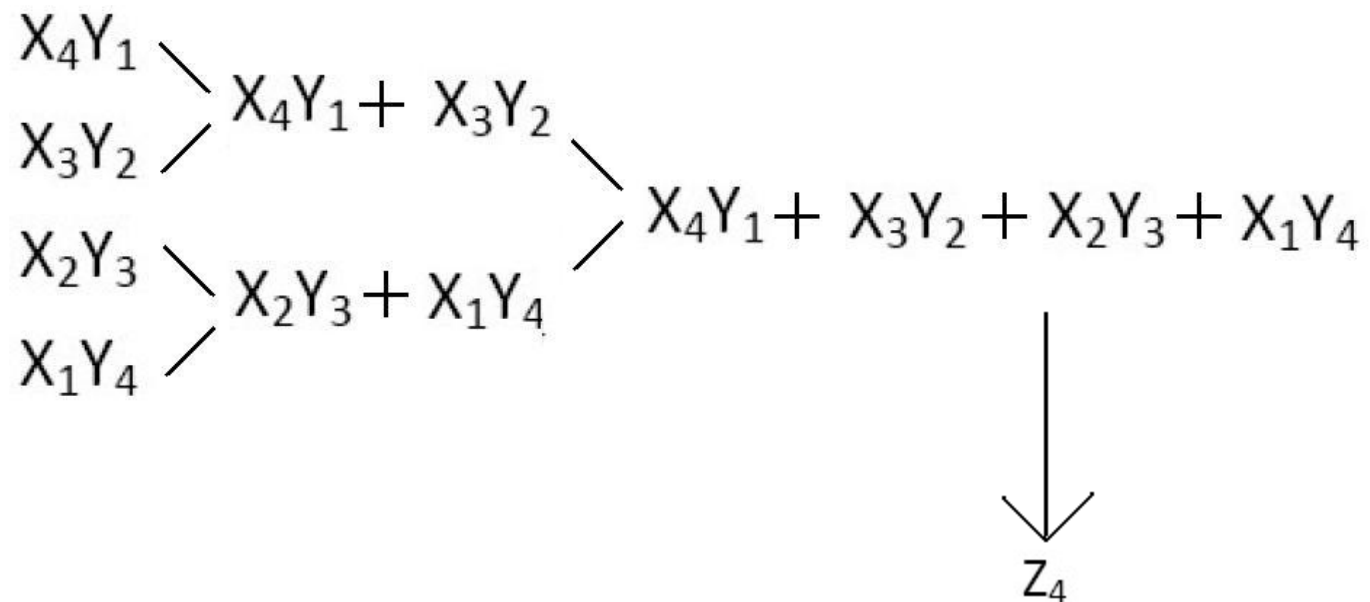
$$X_2 Y_3$$

$$X_1 Y_4$$

Parallel multiplication

$$\begin{array}{l} X_4Y_1 \\ X_3Y_2 \end{array} \begin{array}{l} \backslash \\ / \end{array} X_4Y_1 + X_3Y_2$$
$$\begin{array}{l} X_2Y_3 \\ X_1Y_4 \end{array} \begin{array}{l} \backslash \\ / \end{array} X_2Y_3 + X_1Y_4$$

Parallel multiplication



Parallel modular exponentiation

- Binary modular exponentiation
- Example: $g^{45} \bmod m$

i	2^i	e_i	$g^{2^i} \bmod m$	
0	2^0	1	$g^1 \bmod m$	*
1	2^1	0	$g^2 \bmod m$	
2	2^2	1	$g^4 \bmod m$	*
3	2^3	1	$g^8 \bmod m$	*
4	2^4	0	$g^{16} \bmod m$	
5	2^5	1	$g^{32} \bmod m$	*

$$\begin{aligned} g^{45} \bmod m &= (g^1 \bmod m)(g^4 \bmod m)(g^8 \bmod m)(g^{32} \bmod m) \\ &= g^{(1+4+8+32)} \bmod m = g^{45} \bmod m \end{aligned}$$

Parallel modular exponentiation

- Example: $g^{45} \bmod m$

i	2^i	e_i
0	2^0	1
1	2^1	0
2	2^2	1
3	2^3	1
4	2^4	0
5	2^5	1

Parallel modular exponentiation

- Example: $g^{45} \bmod m$

i	2^i	e_i	$g^{2^i} \bmod m$
0	2^0	1	$g^1 \bmod m$
1	2^1	0	$g^2 \bmod m$
2	2^2	1	$g^4 \bmod m$
3	2^3	1	$g^8 \bmod m$
4	2^4	0	$g^{16} \bmod m$
5	2^5	1	$g^{32} \bmod m$

Parallel modular exponentiation

- Example: $g^{45} \bmod m$

i	2^i	e_i	$g^{2^i} \bmod m$	
0	2^0	1	$g^1 \bmod m$	*
1	2^1	0	$g^2 \bmod m$	
2	2^2	1	$g^4 \bmod m$	*
3	2^3	1	$g^8 \bmod m$	*
4	2^4	0	$g^{16} \bmod m$	
5	2^5	1	$g^{32} \bmod m$	*

Parallel modular exponentiation

- Example: $g^{45} \bmod m$

i	2^i	e_i	$g^{2^i} \bmod m$	
0	2^0	1	$g^1 \bmod m$	*
1	2^1	0	$g^2 \bmod m$	
2	2^2	1	$g^4 \bmod m$	*
3	2^3	1	$g^8 \bmod m$	*
4	2^4	0	$g^{16} \bmod m$	
5	2^5	1	$g^{32} \bmod m$	*

$$\begin{aligned} g^{45} \bmod m &= (g^1 \bmod m)(g^4 \bmod m)(g^8 \bmod m)(g^{32} \bmod m) \\ &= g^{(1+4+8+32)} \bmod m = g^{45} \bmod m \end{aligned}$$

Parallel modular exponentiation

Step i	2^i	e_i	A (α)	B (β)
0	2^0	1	$g^1 \bmod m$	waiting
1	2^1	0	$g^2 \bmod m$	$g^1 \bmod m$
2	2^2	1	$g^4 \bmod m$	waiting
3	2^3	1	$g^8 \bmod m$	$(g^1 g^4) \bmod m = g^5 \bmod m$
4	2^4	0	$g^{16} \bmod m$	$(g^5 g^8) \bmod m = g^{13} \bmod m$
5	2^5	1	$g^{32} \bmod m$	waiting
6				$(g^{13} g^{32}) \bmod m = g^{45} \bmod m$

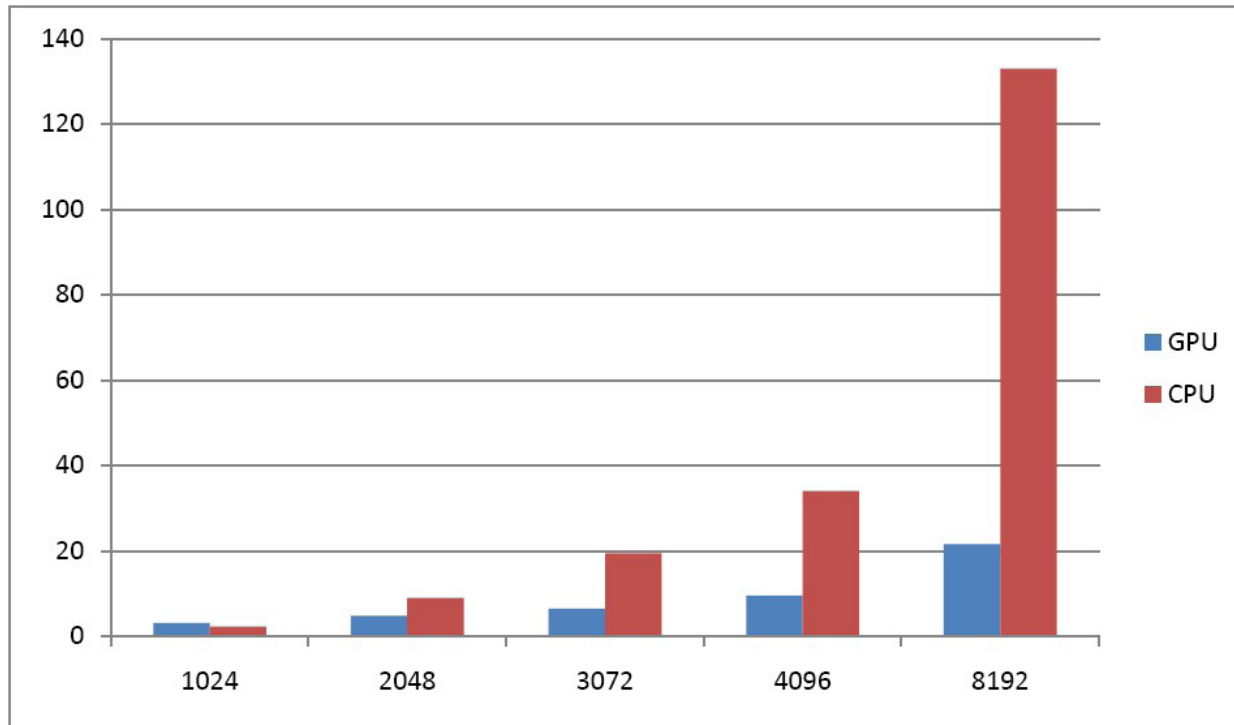
Experimental equipments

- CUDA-enabled GPU
 - NVIDIA GeForce GT 130M
 - 1.5 GHz with 32 cores
- CPU
 - Intel Core2 Duo P8700 processor 2.53 GHz

Performance Comparison

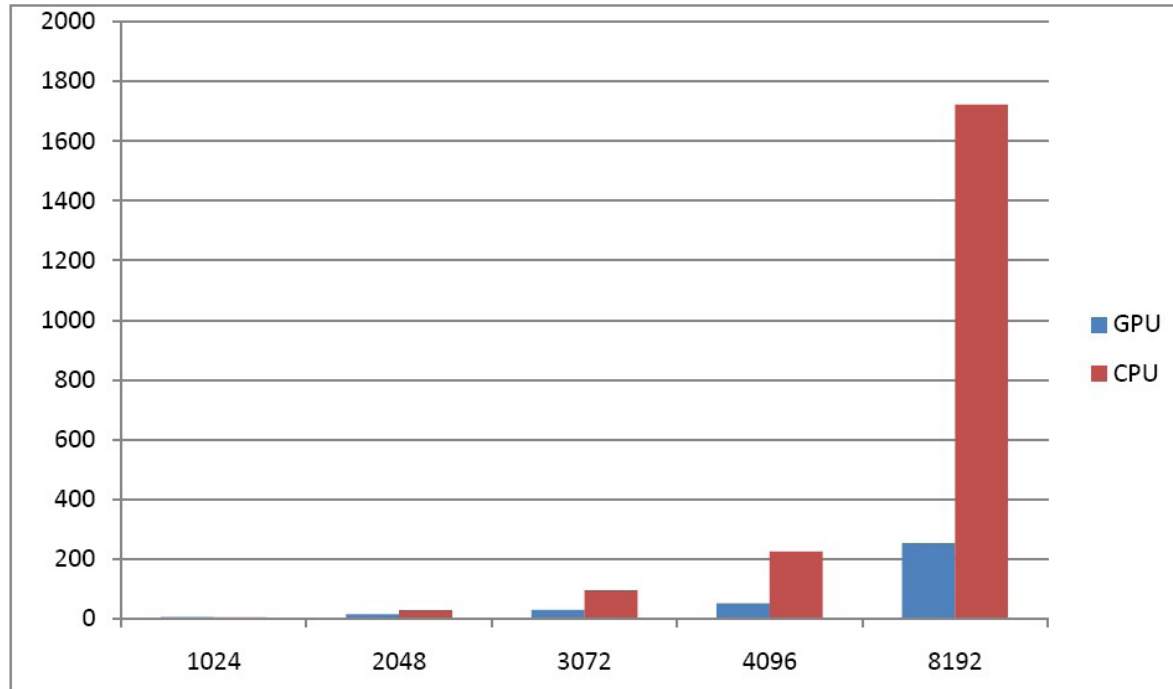
- 1.5 GHz with 32 cores vs. 2.53GHz
- Bit-lengths with 100 groups input data each
 - 1024 bits
 - 2048 bits
 - 3072 bits
 - 4096 bits
 - 8192 bits

Modular multiplication



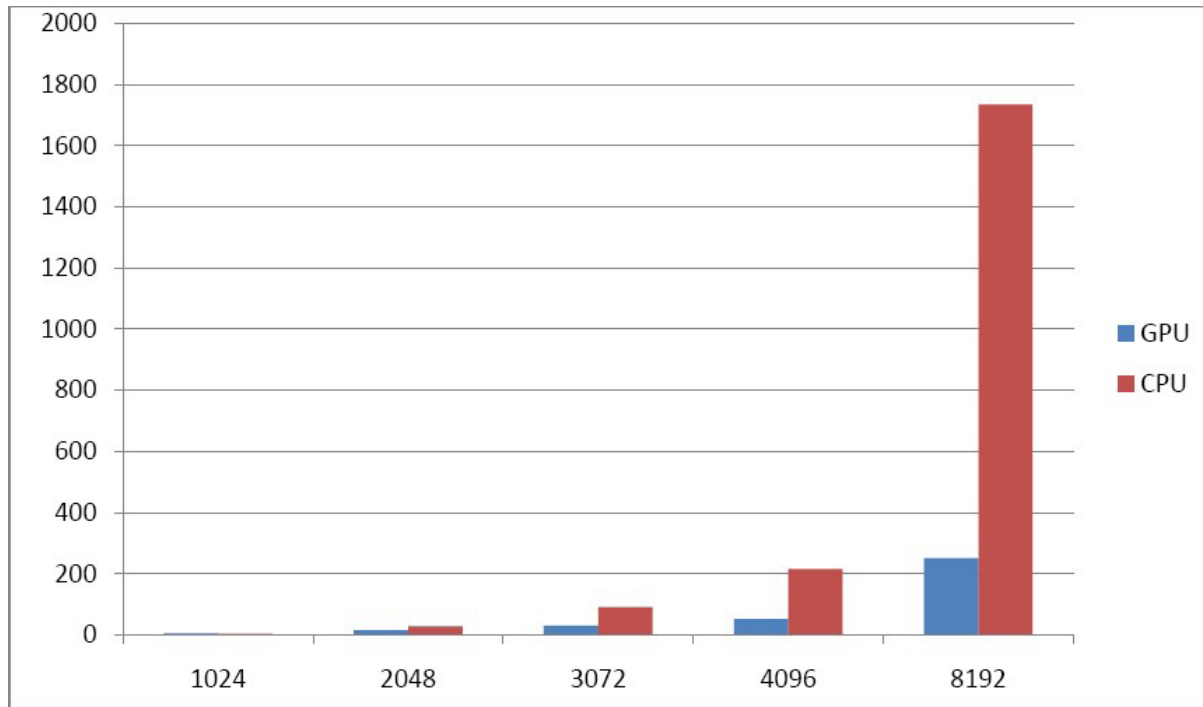
- 1.5 GHz with 32 cores vs. 2.53GHz
- Max 6.14x speedup

Modular exponentiation



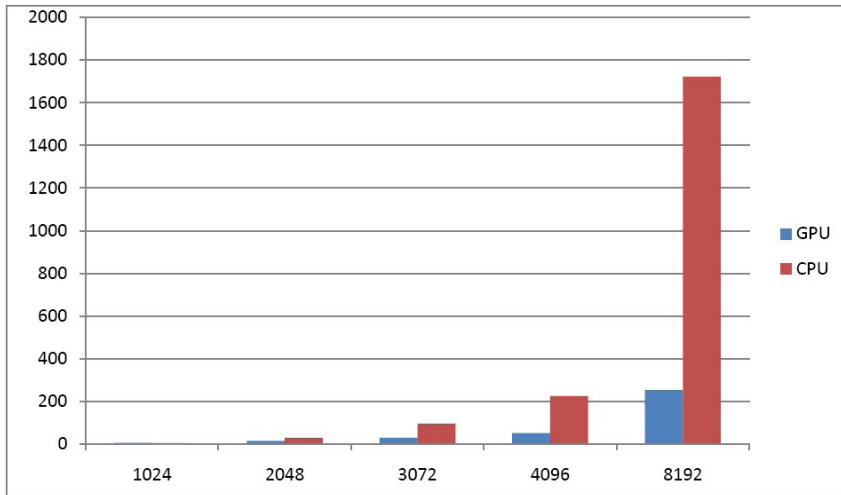
- 1.5 GHz with 32 cores vs. 2.53GHz
- Max 6.82x speedup

RSA

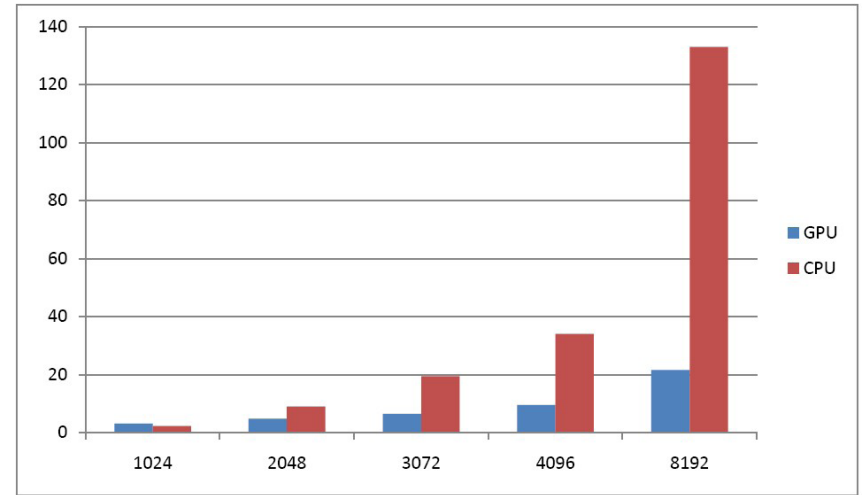


- 1.5 GHz with 32 cores vs. 2.53GHz
- Max 6.92x speedup

Modular exponentiation



Modular multiplication



Max 6.82x speedup

- Not be improved significantly
- Limited processors

Max 6.14x speedup

Discussion and conclusion

- Factors influence the CUDA's performance
 - Some sequential methods
 - Access speeds of various memories
 - Communication
 - Synchronization

Discussion and conclusion

- Public key algorithms can be implemented fast on a CUDA-enabled GPU
- CPU implementation
 - The n -by- n multiplies will take four times as long, and the exponent is doubled if bit-length is doubled.
 - Execution time of RSA will take 8 times as long.
- GPU implementation
 - execution time will take 4-5 times as long if the bit-length is doubled.

Question?