

# High-Performance Modular Multiplication on the Cell Broadband Engine

Joppe W. Bos

Laboratory for Cryptologic Algorithms  
EPFL, Lausanne, Switzerland  
[joppe.bos@epfl.ch](mailto:joppe.bos@epfl.ch)



- Motivation and previous work
- Applications for multi-stream modular multiplication
- Background
  - Fast reduction with special primes
  - The Cell broadband engine
- Modular multiplications on the Cell
- Performance results
- Conclusions

Modular multiplication is a performance critical operation in many cryptographic applications

- RSA
- ElGamal
- Elliptic curve cryptosystems

as well as in cryptanalytic applications

- computing elliptic curve discrete logarithms (Pollard rho)
- factoring integers (elliptic curve factorization method)

**Measure the performance on the Cell.**

## Misc. Platforms

Lots of performance results for many platforms

- GNU Multiple Precision (GMP) Arithmetic Library: almost all platforms (multiplication + reduction separately),
- Bernstein et. al (Eurocrypt 2009): NVIDIA GPUs,
- Brown et al. (CT-RSA 2001): NIST primes on x<sub>86</sub>.

## On the Cell Broadband Engine

- The Multi-Precision Math (MPM) Library by IBM,

Optimize for one specific bit-size

- Costigan and Schwabe (Africacrypt 2009): special 255-bit prime,
- Bernstein et. al (SHARCS 2009): 195-bit generic moduli

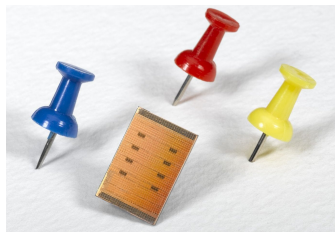
# Contributions

## What did I do?

Present high-performance multi-stream algorithms

- Montgomery multiplication,
- schoolbook multiplication,
- various special reduction schemes.

Implementation details (in C) are presented for a cryptologic interesting range **192 – 521** bits targeted at the **Cell Broadband Engine**.



# Multi-Stream Modular Multiplication Applications

Modular exponentiations using a square-and-multiply algorithm.

## Cryptography

- Exponentiations with the same random exponent:
  - ElGamal encryption (ElGamal, Crypto 1984),
  - Double base ElGamal - Damgård ElGamal (Damgård, Crypto 1991),
  - “Double” hybrid Damgård ElGamal (Kiltz et. al, Eurocrypt 2009).
- Batch decryption in elliptic curve cryptosystems

## Cryptanalysis

- Pollard rho (elliptic curve discrete logarithm problem)
- Integer factorization (elliptic curve factorization method)

Faster reduction exploiting the structure of the special prime.

## By US National Institute of Standards

Five recommended primes in the FIPS 186-3 (Digital Signature Standard)

$$P_{192} = 2^{192} - 2^{64} - 1$$

$$P_{224} = 2^{224} - 2^{96} + 1$$

$$P_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$

$$P_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$$

$$P_{521} = 2^{521} - 1$$

## Prime used in Curve25519

Proposed by Bernstein at PKC 2006

$$P_{255} = 2^{255} - 19$$

Example:  $P_{192} = 2^{192} - 2^{64} - 1$

$$0 \leq x < P_{192}^2, \quad 0 \leq x_H, x_L < 2^{192}, \quad x = x_H \cdot 2^{192} + x_L$$

$$x \equiv x_L + x_H \cdot 2^{64} + x_H \pmod{P_{192}}$$



Example:  $P_{192} = 2^{192} - 2^{64} - 1$

$$0 \leq x < P_{192}^2, \quad 0 \leq x_H, x_L < 2^{192}, \quad x = x_H \cdot 2^{192} + x_L$$

$$x \equiv x_L + x_H \cdot 2^{64} + x_H \pmod{P_{192}}$$
$$x_H \cdot 2^{64} < 2^{256}$$

$$x_H \cdot 2^{64} \equiv x_H \cdot 2^{64} \pmod{2^{192}} + \left\lfloor \frac{x_H \cdot 2^{64}}{2^{192}} \right\rfloor \cdot 2^{64} + \left\lfloor \frac{x_H \cdot 2^{64}}{2^{192}} \right\rfloor \pmod{P_{192}}$$

Example:  $P_{192} = 2^{192} - 2^{64} - 1$

$$0 \leq x < P_{192}^2, \quad 0 \leq x_H, x_L < 2^{192}, \quad x = x_H \cdot 2^{192} + x_L$$

$$x \equiv x_L + x_H \cdot 2^{64} + x_H \pmod{P_{192}}$$
$$x_H \cdot 2^{64} < 2^{256}$$

$$x_H \cdot 2^{64} \equiv x_H \cdot 2^{64} \pmod{2^{192}} + \left\lfloor \frac{x_H \cdot 2^{64}}{2^{192}} \right\rfloor \cdot 2^{64} + \left\lfloor \frac{x_H \cdot 2^{64}}{2^{192}} \right\rfloor \pmod{P_{192}}$$

$$s_1 = (c_5, c_4, c_3, c_2, c_1, c_0), \quad s_2 = (c_{11}, c_{10}, c_9, c_8, c_7, c_6),$$
$$s_3 = (c_9, c_8, c_7, c_6, 0, 0), \quad s_4 = (0, 0, c_{11}, c_{10}, 0, 0),$$
$$s_5 = (0, 0, 0, 0, c_{11}, c_{10}) \quad \text{Return } s_1 + s_2 + s_3 + s_4 + s_5$$

Example:  $P_{192} = 2^{192} - 2^{64} - 1$

$$0 \leq x < P_{192}^2, \quad 0 \leq x_H, x_L < 2^{192}, \quad x = x_H \cdot 2^{192} + x_L$$

$$x \equiv x_L + x_H \cdot 2^{64} + x_H \pmod{P_{192}}$$
$$x_H \cdot 2^{64} < 2^{256}$$

$$x_H \cdot 2^{64} \equiv x_H \cdot 2^{64} \pmod{2^{192}} + \left\lfloor \frac{x_H \cdot 2^{64}}{2^{192}} \right\rfloor \cdot 2^{64} + \left\lfloor \frac{x_H \cdot 2^{64}}{2^{192}} \right\rfloor \pmod{P_{192}}$$

$$s_1 = (c_5, c_4, c_3, c_2, c_1, c_0), \quad s_2 = (0, 0, c_7, c_6, c_7, c_6),$$
$$s_3 = (c_9, c_8, c_9, c_8, 0, 0), \quad s_4 = (c_{11}, c_{10}, c_{11}, c_{10}, c_{11}, c_{10})$$

Return  $s_1 + s_2 + s_3 + s_4$

Solinas, technical report 1999

Note: this reduces to  $[0, 4 \cdot P_{192}]$

# Multiplication algorithms

## Generic Moduli

Montgomery multiplication (with final subtraction)

## Special Moduli

Multiplication + special reduction

Size of the modulus: 192 - 521 bit

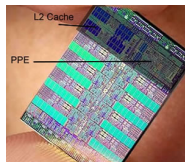
Multiplication method: schoolbook

Investigate other methods (such as Karatsuba) is left as future work.

# The Cell Broadband Engine

Cell architecture in the PlayStation 3 (@ 3.2 GHz):

- Broadly available (24.6 million)
- Relatively cheap (US\$ 300)



The Cell contains

- eight **"Synergistic Processing Elements"** (SPEs)  
six available to the user in the PS3
- one "Power Processor Element" (PPE)
- the Element Interconnect Bus (EIB)  
a specialized high-bandwidth circular data bus



The SPEs contain

- a Synergistic Processing Unit (SPU)
  - Access to 128 registers of 128-bit
  - SIMD operations
  - Dual pipeline (odd and even)
  - Rich instruction set
  - In-order processor
- 256 KB of fast local memory (Local Store)
- Memory Flow Controller (MFC)

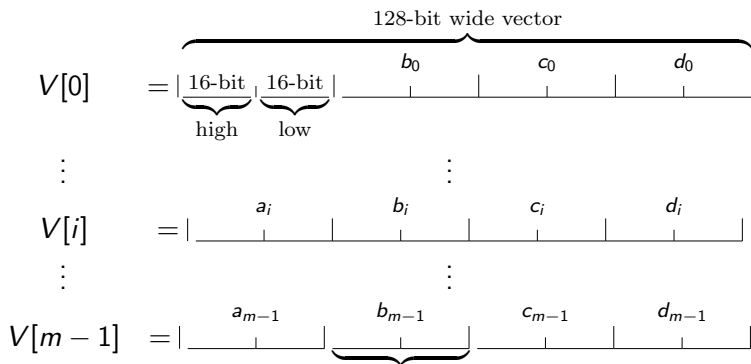
# Programming Challenges

- Memory
  - The executable **and** all data should fit in the LS
  - *Or* perform manual DMA requests to the main memory (max. 214 MB)
- Branching
  - No “smart” dynamic branch prediction
  - Instead “prepare-to-branch” instructions to redirect instruction prefetch to branch targets
- Instruction set limitations
  - $16 \times 16 \rightarrow 32$  bit multipliers (4-SIMD)
- Dual pipeline
  - One odd and one even instruction can be dispatched per clock cycle.

# Modular Multiplication on the Cell I

Four  $(16 \cdot m)$ -bit integers  $A, B, C, D$  represented in  $m$  vectors.

$$X = \sum_{i=0}^{m-1} x_i \cdot 2^{16 \cdot i}$$



the most significant position of  $X_1$  is located in  
either the lower or higher 16-bit of the 32-bit word



## Implementation

- use the multiply-and-add instruction,
    - if  $0 \leq a, b, c, d < 2^{16}$ , then  $a \cdot b + c + d < 2^{32}$ .
  - try to fill both the odd and even pipelines,
  - are branch-free.
- 
- Do not fully reduce modulo ( $m$ -bits)  $P$ ,
  - Montgomery and special reduction  $[0, 2^m)$ ,
  - These numbers can be used as input again,
  - Reduce to  $[0, P)$  at the cost of a single comparison + subtraction.

# Modular Multiplication on the Cell III

Special reduction  $\rightarrow [0, t \cdot P)$  ( $t \in \mathbb{Z}$  and small)

How to reduce to  $[0, 2^m)$ ?

- Apply special reduction again
- Repeated subtraction ( $t$  times)

For a constant modulus  $m$ -bit  $P$

Select the four values to subtract simultaneously using `select` and `cmpgt` instructions and a look-up table.

# Modular Multiplication on the Cell IV

For the special primes this can be done even faster.

| t | $t \cdot P_{224} = t \cdot (2^{224} - 2^{96} + 1) = \{c_7, \dots, c_0\}$ |              |              |              |              |       |       |       |
|---|--|--------------|--------------|--------------|--------------|-------|-------|-------|
|   | $c_7$  | $c_6$        | $c_5$        | $c_4$        | $c_3$        | $c_2$ | $c_1$ | $c_0$ |
| 0 | 0  | 0            | 0            | 0            | 0            | 0     | 0     | 0     |
| 1 | 0  | $2^{32} - 1$ | $2^{32} - 1$ | $2^{32} - 1$ | $2^{32} - 1$ | 0     | 0     | 1     |
| 2 | 1  | $2^{32} - 1$ | $2^{32} - 1$ | $2^{32} - 1$ | $2^{32} - 2$ | 0     | 0     | 2     |
| 3 | 2  | $2^{32} - 1$ | $2^{32} - 1$ | $2^{32} - 1$ | $2^{32} - 3$ | 0     | 0     | 3     |
| 4 | 3  | $2^{32} - 1$ | $2^{32} - 1$ | $2^{32} - 1$ | $2^{32} - 4$ | 0     | 0     | 4     |

- $c_0 = t$ ,  $c_1 = c_2 = 0$  and  $c_3 = (\text{unsigned int}) (0 - t)$ .
- If  $t > 0$  then  $c_4 = c_5 = c_6 = 2^{32} - 1$  else  $c_4 = c_5 = c_6 = 0$ .
- Use a single select.

# Modular Multiplication on the Cell V

$$P_{255} = 2^{255} - 19$$

## Original approach

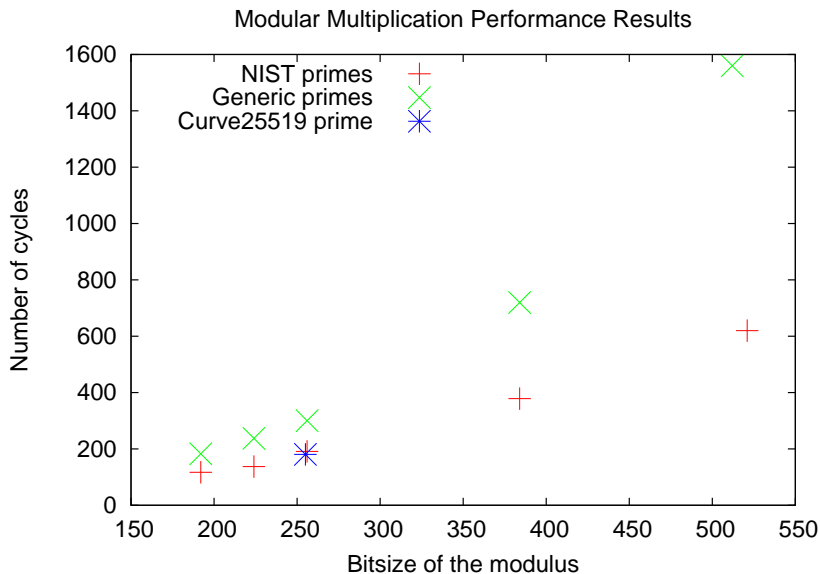
Proposed by Bernstein and implemented on the SPE by Costigan and Schwabe (Africacrypt 2009):

Here  $x \in \mathbb{F}_{2^{255}-19}$  is represented as  $x = \sum_{i=0}^{19} x_i 2^{\lceil 12.75i \rceil}$ .

## Redundant representation

- Following ideas from Bos, Kaihara and Montgomery (SHARCS 2009),
- Calculate modulo  $2 \cdot P_{255} = 2^{256} - 38 = \sum_{i=0}^{15} x_i 2^{16}$ ,
- Reduce to  $[0, 2^{256})$ .

# Performance Results



$$\frac{\text{Montgomery multiplication}}{\text{multiplication} + \text{fast reduction}} \approx 1.4 - 2.5$$

# Comparison Special Moduli

## Number of cycles for what?

- Measurements over millions of multi-stream modular multiplications,
- Cycles for a single modular multiplication,
- include benchmark overhead, function call, loading (storing) the input (output), converting from radix- $2^{32}$  to radix- $2^{16}$ .

# Comparison Special Moduli

## Number of cycles for what?

- Measurements over millions of multi-stream modular multiplications,
- Cycles for a single modular multiplication,
- include benchmark overhead, function call, loading (storing) the input (output), converting from radix-2<sup>32</sup> to radix-2<sup>16</sup>.

## Special prime $P_{255}$

- Costigan and Schwabe (Africacrypt 2009), 255 bit.
- single-stream: 444 cycles (144 mul, 244 reduction, 56 overhead).
- multi-stream: 168 cycles.
  - no function call, loading and storing,  
“perfectly” scheduled (filled both pipelines)
- this work: 180 cycles ( $< 168 + 56$ ),
- both approaches are comparable in terms of speed (on the Cell).

# Comparison Generic Moduli

## Generic 195-bit moduli

- Bernstein et al. (SHARCS 2009), multi-stream, 189 cycles,
- This work: multi-stream, 159 cycles for 192-bit generic moduli,
- Scaling:  $(\frac{195}{192})^2 \cdot 159 = 164$  cycles.

## Generic moduli

| Bitsize | #cycles |       |       |
|---------|---------|-------|-------|
|         | New     | MPM   | uMPM  |
| 192     | 159     | 1,188 | 877   |
| 224     | 237     | 1,188 | 877   |
| 256     | 300     | 1,188 | 877   |
| 384     | 719     | 2,092 | 1,610 |
| 512     | 1,560   | 3,275 | 2,700 |



# Conclusions

- We presented SIMD algorithms for Montgomery and schoolbook multiplication and fast reduction.
- Implementations are optimized for the Cell architecture.
- Implementation results for moduli of size 192 to 521 bits show that special primes are 1.4 to 2.5 times faster compared to generic primes.

## Future work

- Try Karatsuba multiplication
- Further optimize Montgomery multiplication (almost finished)