High-Performance Modular Multiplication on the Cell Processor

Joppe W. Bos

Laboratory for Cryptologic Algorithms
EPFL, Lausanne, Switzerland
joppe.bos@epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Outline

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

Motivation

Modular multiplication is time-critical in cryptography

- RSA (≥ 2048 bits)
- ElGamal (≥ 2048 bits)
- ECC (≥ 224 bits)

and in cryptanalysis

- ECDLP ($\approx 100 - 200$ bits)
- ECM (≈ 200 bits)

Motivation

Multi-stream **modular multiplication is time-critical in cryptography**

- RSA (≥ 2048 bits)
- ElGamal (≥ 2048 bits)
  - 2, ElGamal encryption (ElGamal, CRYP84)
  - 3, Damgård ElGamal (Damgård, CRYP91)
  - 4, “Double” hybrid Damgård ElGamal (Kiltz et al., EUR09)
- ECC (≥ 224 bits)
  - ∞, Batch decryption PSEC, ECIES

and in cryptanalysis

- ECDLP ($\approx 100 - 200$ bits)
  - ∞, Pollard ρ
- ECM (≈ 200 bits)
  - ∞, different curves

Previous works

Misc. Platforms

Lots of performance results for many platforms

- GNU Multiple Precision (GMP) Arithmetic Library
  many platforms, **no Montgomery multiplication**
- Bernstein et al. (EUR09): NVIDIA GPUs
- Brown et al. (CT-RSA01): NIST primes on x86

On the Cell Broadband Engine

Optimize for one specific bit-size

- The Multi-Precision Math (MPM) Library by IBM (single stream)
- Costigan and Schwabe (AFR09): special 255-bit prime (multi-stream)
- Bernstein et al. (SHARCS09): 195-bit generic moduli (multi-stream)

Contributions




- Fast algorithms for modular multiplication on platforms with a **multiply-and-add**.
- Target a **range** of moduli: 192 – 521 bits.
- Compare **generic** and **special** modular multiplication
- All implementations set new **speed records** for the Cell Broadband Engine

How much faster is using special moduli compared to generic ones?

Special Primes

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)

$$
\begin{aligned}
P_{192} &= 2^{192} - 2^{64} - 1 \\
P_{224} &= 2^{224} - 2^{96} + 1 \\
P_{256} &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\
P_{384} &= 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\
P_{521} &= 2^{521} - 1
\end{aligned}
$$

Prime used in Curve25519

Proposed by Bernstein at PKC 2006

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

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

$$0 \le x < P_{192}^2$$

$$\begin{aligned} x &= x_H \cdot 2^{192} + x_L \\ &\equiv x_H \cdot 2^{192} + x_L - x_H \cdot P_{192} \bmod P_{192} \\ &= x_L + x_H \cdot 2^{64} + x_H \end{aligned}$$

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

$$
\begin{aligned}
& 0 \le x < P_{192}^2 \\
x &= x_H \cdot 2^{192} + x_L \\
&\equiv x_H \cdot 2^{192} + x_L - x_H \cdot P_{192} \bmod P_{192} \\
&= x_L + {\color{red} x_H \cdot 2^{64}} + x_H
\end{aligned}
$$

$$
\begin{aligned}
{\color{red} x_H \cdot 2^{64}} &= \left\lfloor \frac{x_H \cdot 2^{64}}{2^{192}} \right\rfloor \cdot 2^{192} + (x_H \cdot 2^{64} \bmod P_{192}) \\
&= \tilde{x}_H \cdot 2^{192} + \tilde{x}_L \\
&\equiv \tilde{x}_H \cdot 2^{192} + \tilde{x}_L - \tilde{x}_H \cdot P_{192} \bmod P_{192} \\
&= \tilde{x}_L + \tilde{x}_H \cdot 2^{64} + \tilde{x}_H
\end{aligned}
$$

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

$$0 \le x < P_{192}^2$$

$$\begin{aligned} x &= x_H \cdot 2^{192} + x_L \\ &\equiv x_H \cdot 2^{192} + x_L - x_H \cdot P_{192} \bmod P_{192} \\ &= x_L + {\color{red} x_H \cdot 2^{64}} + x_H \end{aligned}$$

$$\begin{aligned} {\color{red} x_H \cdot 2^{64}} &= \left\lfloor \frac{x_H \cdot 2^{64}}{2^{192}} \right\rfloor \cdot 2^{192} + (x_H \cdot 2^{64} \bmod P_{192}) \\ &= \tilde{x}_H \cdot 2^{192} + \tilde{x}_L \\ &\equiv \tilde{x}_H \cdot 2^{192} + \tilde{x}_L - \tilde{x}_H \cdot P_{192} \bmod P_{192} \\ &= \tilde{x}_L + \tilde{x}_H \cdot 2^{64} + \tilde{x}_H \end{aligned}$$

$$x = (c_{11}, \ldots, c_0)$$

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

$s_3 = (c_9, c_8, c_7, c_6, 0, 0)$, $s_4 = (0, 0, c_{11}, c_{10}, 0, 0)$,

$s_5 = (0, 0, 0, 0, c_{11}, c_{10})$ Return $s_1 + s_2 + s_3 + s_4 + s_5$

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

$$0 \leq x < P_{192}^2$$

$$\begin{aligned} x &= x_H \cdot 2^{192} + x_L \\ &\equiv x_H \cdot 2^{192} + x_L - x_H \cdot P_{192} \bmod P_{192} \\ &= x_L + x_H \cdot 2^{64} + x_H \end{aligned}$$

$$\begin{aligned} x_H \cdot 2^{64} &= \left\lfloor \frac{x_H \cdot 2^{64}}{2^{192}} \right\rfloor \cdot 2^{192} + (x_H \cdot 2^{64} \bmod P_{192}) \\ &= \tilde{x}_H \cdot 2^{192} + \tilde{x}_L \\ &\equiv \tilde{x}_H \cdot 2^{192} + \tilde{x}_L - \tilde{x}_H \cdot P_{192} \bmod P_{192} \\ &= \tilde{x}_L + \tilde{x}_H \cdot 2^{64} + \tilde{x}_H \end{aligned}$$

$$x = (c_{11}, \ldots, c_0)$$

$$s_1 = (c_5, c_4, c_3, c_2, c_1, c_0), \quad s_2 = (0, 0, c_7, c_6, c_7, c_6),$$

$$s_3 = (c_9, c_8, c_9, c_8, 0, 0), \quad s_4 = (c_{11}, c_{10}, c_{11}, c_{10}, c_{11}, c_{10})$$

$$\text{Return } 0 \leq s_1 + s_2 + s_3 + s_4 < 4 \cdot P_{192}$$

Solinas, technical report 1999

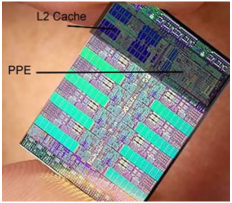
The Cell Broadband Engine

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

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

PS3 Slim or the newest firmware disables another OS!

Also available in blade servers and PCI express boards.



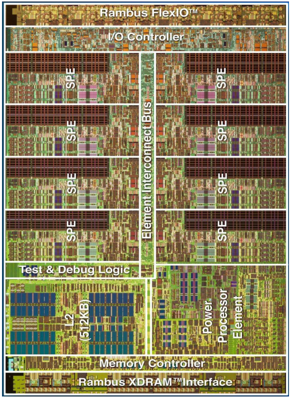
The Cell contains

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

Cell architecture, the SPEs

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)

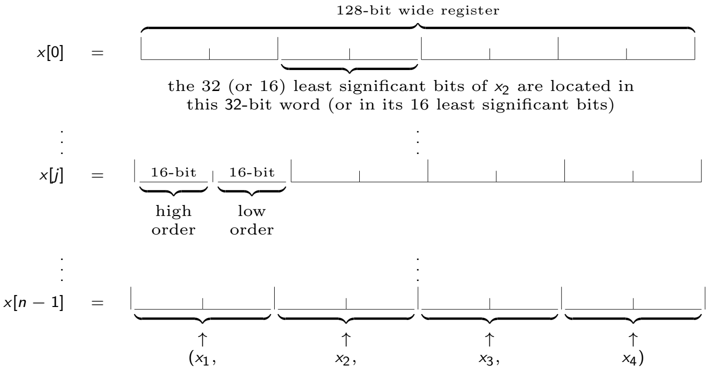


Programming Challenges

- 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 in m vectors: $x_i = \sum_{j=0}^{m-1} x_{i,j} \cdot 2^{16 \cdot j}$



Modular Multiplication on the Cell II

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 $[0, 2^m\rangle$,
- These numbers can be used as input again,
- Reduce to $[0, P\rangle$ at the cost of a single comparison + subtraction.

Modular Multiplication on the Cell III

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

How to reduce to $[0, 2^m\rangle$? $(2^{m-1} < P < 2^m)$

- Apply special reduction again
- Repeated subtraction ($(t-1)$ 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	$t \cdot P_{224} = t \cdot \left(2^{224} - 2^{96} + 1\right) = \{c_7, \ldots, c_0\}$							
	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

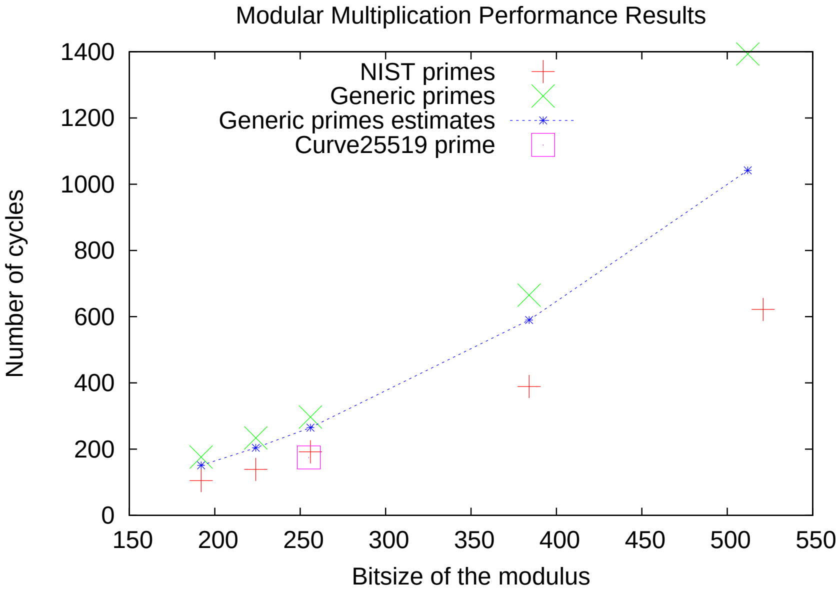
Proposed by Bernstein and implemented on the SPE by Costigan and Schwabe (Africacrypt 2009):

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 $[0, 2^{256}\rangle$.

Performance Results



$\frac{\text{Montgomery multiplication}}{\text{multiplication + fast reduction}} \approx 1.4 - 1.7$, (512 bits: 2.2 times faster)

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^{32} to radix-2^{16}.

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, multi-stream: 175 cycles ($< 168 + 56$),
- $\Longrightarrow$ 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, 176 cycles (for 192-bit generic moduli)
  Scale: $(\frac{195}{192})^2 \cdot 176 = 182$ cycles.

Generic moduli: single vs. multi stream

Bitsize	#cycles		
	New	MPM	uMPM
192	176	1,188	877
224	234	1,188	877
256	297	1,188	877
384	665	2,092	1,610
512	1,393	3,275	2,700

Conclusions

- We presented SIMD algorithms for Montgomery and { schoolbook, Karatsuba } multiplication plus fast reduction.
- Algorithms are optimized for architectures with a multiply-and-add instruction.
- Implementation results on the Cell: moduli of size 192 to 521 bits show that special primes are at least 1.4 times faster compared to generic primes.

Future work

- Similair speed-up on other multi-core platforms like GPUs?