Performances de schémas d'évaluation polynomiale sur architectures vectorielles

8ème Rencontres Arithmétique de l'Informatique Mathématique
Banyuls

Hugues de Lassus Saint-Geniès et Guillaume Revy

DALI, Université de Perpignan Via Domitia
LIRMM, Université de Montpellier
CNRS UMR 5506

28 – 30 juin 2016

Motivations

Approximation of **elementary functions**

- Tedious, architecture-dependant and **error-prone** implementations
- Wide variety of software libraries: **performances** vs **accuracy**
- Often involves **polynomial evaluations**

Goals of the MetaLibm ANR project:

- Automatize **code generation** for mathematical functions and filters
- Get **high performances** given a target accuracy and architecture

Our goals:

- Generate $\begin{cases}\textbf{vectorizable} \\ \textbf{vectorized}\end{cases}$ code for polynomial evaluation
- Use features of the target architectures to improve performances

This talk

In this talk:

- How do different polynomial schemes behave on **SIMD FP units**?
- Study of the performance of **classic** and **latency-minimal schemes**
- **Native precision** computations
- No adaptation of coefficients (Knuth & Eve, Paterson & Stockmeyer)
- Impact on **elementary function evaluation** (a vectorized logarithm)

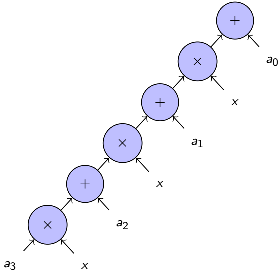
Some conclusions:

- The classic Horner's rule is quickly **not the best choice** for efficiency.
- **Denormalization** can be fatal to performance.
- **Faster**, more **parallel schemes** can have enough accuracy for **faithfully-rounded** functions.

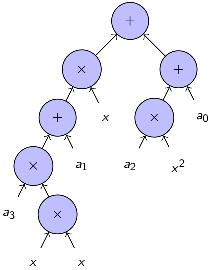
Outline

Outline

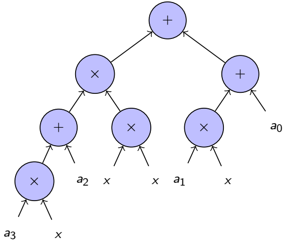
The most usual polynomial schemes:



Horner's rule



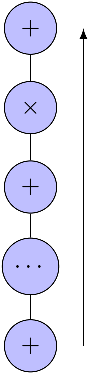
2[nd]-order Horner's rule



Estrin's scheme

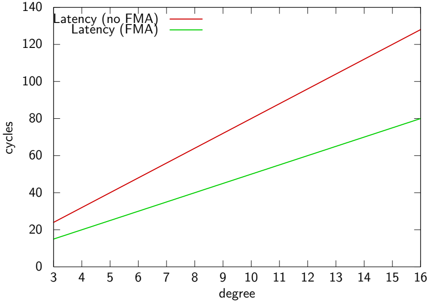
Latencies: Horner's rule

$$\text{Latency} = \#\ \text{cycles required for a complete evaluation}$$
$$= \#\ \text{cycles for the critical path on infinite parallelism}$$

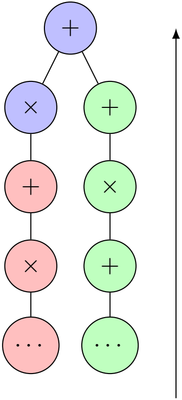


n ADD + n MUL

Evolution of the latency with Haswell costs for MUL, ADD and FMA

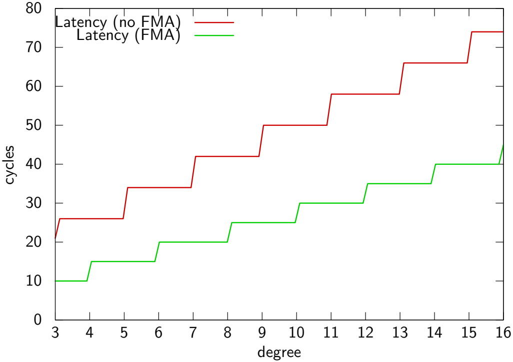


Latencies: Horner-2

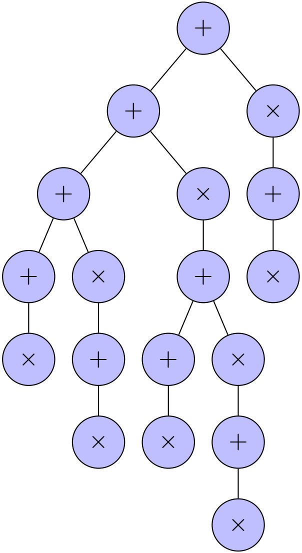


n ADD $+$
$n+1$ MUL

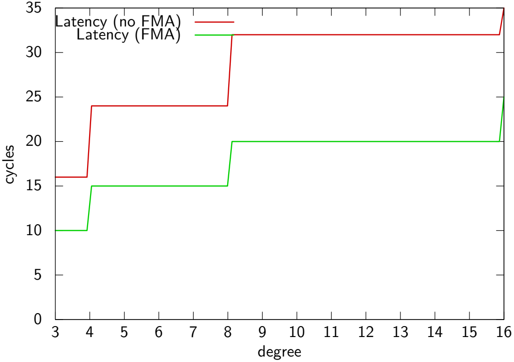
Evolution of the latency with Haswell costs for MUL, ADD and FMA



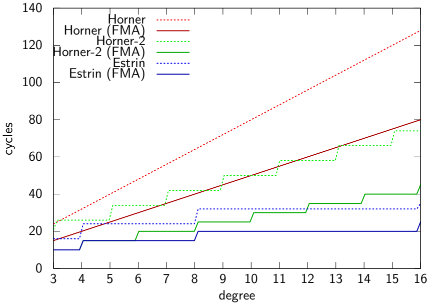
Latencies: Estrin's scheme



Evolution of the latency with Haswell costs for `MUL`, `ADD` and `FMA`

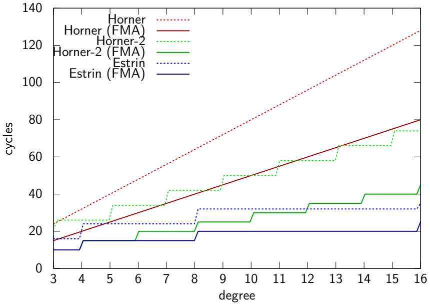


Latencies: first conclusions



On pipelined architectures, passed a certain degree, latencies can be lowered with more parallel schemes.

Latencies: first conclusions



On pipelined architectures, passed a certain degree, latencies can be lowered with more parallel schemes.

How is throughput impacted?

- As intruction parallelism increases, so does the pressure on registers.
- A common "rule of thumb" is: parallel schemes have **better latencies** but **worse throughputs** than sequential schemes.

Outline

Studying the vectorization of polynomial evaluations

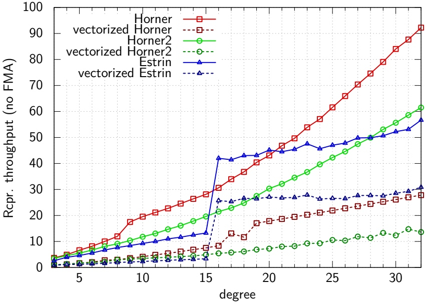
Objectives:

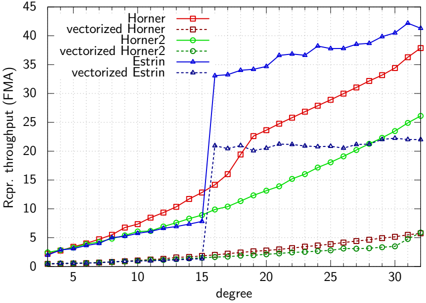
- Be able to choose **fast** polynomial evaluation schemes
- Benefit from **vector extensions** of our ISA (SWAR)

Protocol:

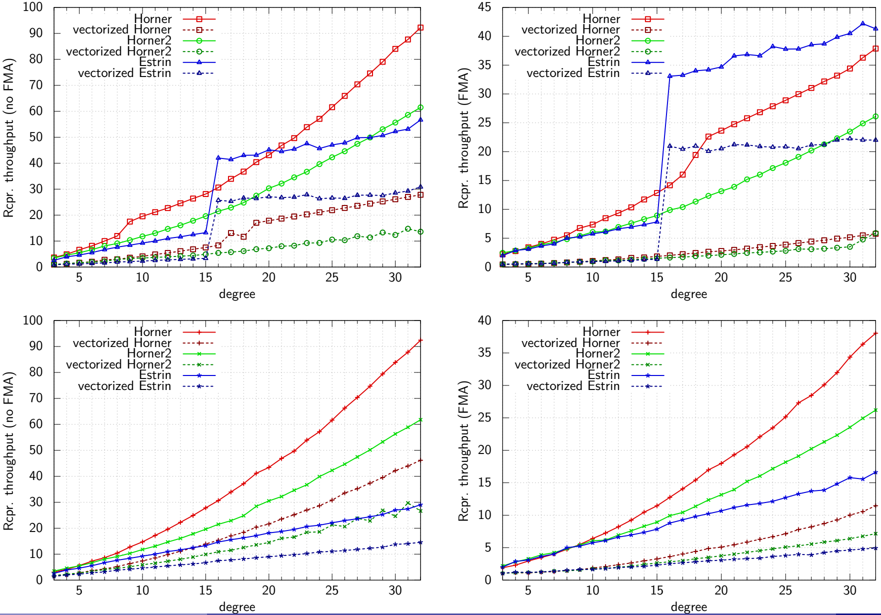
- Benchmarks of **Horner, 2^{nd}-order Horner and Estrin schemes**
- Polynomials of degrees ranging from 3 to 32
- Using Taylor(log, 1) for the coefficients
- ISA featuring **AVX2**, with or without FMA (Haswell, ∼Ivy Bridge)
- Single / double precision
- Compiler: GCC 5.2.0

Scalar vs. autovectorized code: First observations





Scalar vs. autovectorized code: First observations



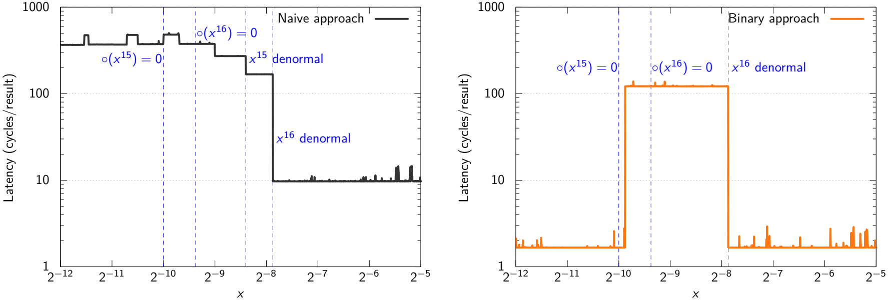
Impact of denormalization (1)

- Do arithmetic instructions always have the same latency, **regardless of the inputs**?
- For instance, what happens when x^n or a partial result is **denormal**?

Experimental protocol:

- 2^{10} samples x between 2^{-12} and 2^{-5}
- Measure the **latency to compute** x^{16} using a naive and a binary approach:
  - naive: $x \otimes x \otimes \cdots \otimes x$
  - binary: $((x \otimes x) \otimes (x \otimes x)) \otimes \ldots$

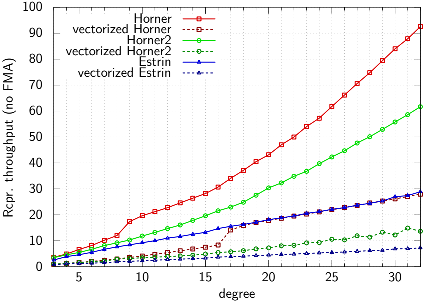
Impact of denormalization (2)

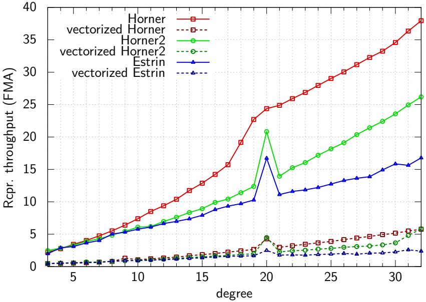


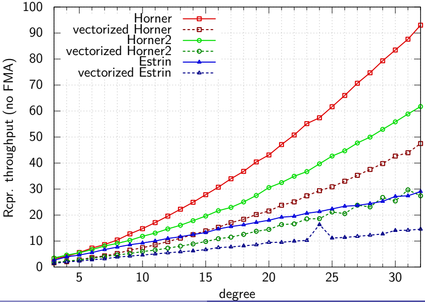
120-150 cycles penalty for `MULSS normal, normal -> denormal`

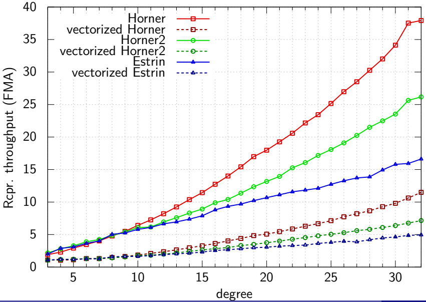
$\rightsquigarrow$ Confirmed by Agner Fog's optimization manuals

Throughputs with GCC 5.2 for inputs in $[1, 2]$









What do these graphs say?

The obvious:

- FMA is a $\approx$**2x-gain**;
- **Parallel schemes** become more efficient as n increases

The not-so-obvious:

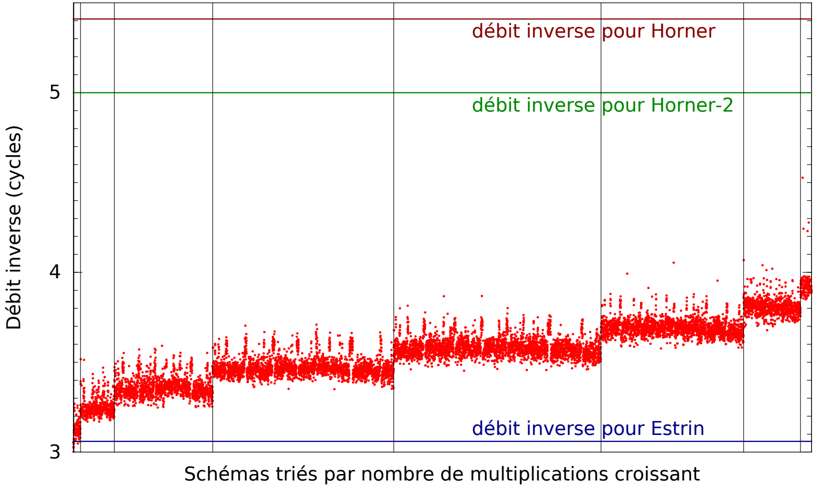
- **Denormalization** can introduce huge overheads;
- The sixteen 256-bit AVX registers seem to resist well the pressure of moderately high degrees;

Are there even **more parallel**, **more efficient** schemes? $\rightarrow$ use CGPE[1]

[1] `http://cgpe.gforge.inria.fr/`

Comparisons with other schemes (1)

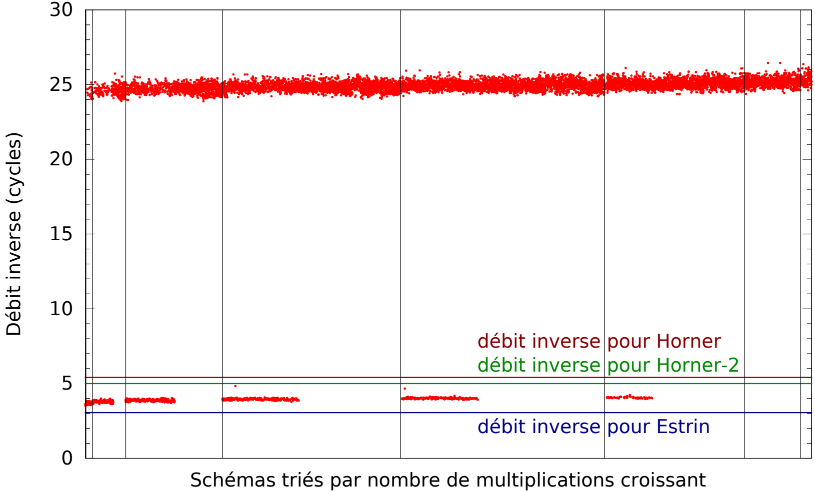
All schemes of minimal latency (for Haswell without FMA) for the degree 12, compared with classic schemes



. . . when no denormalization occurs.

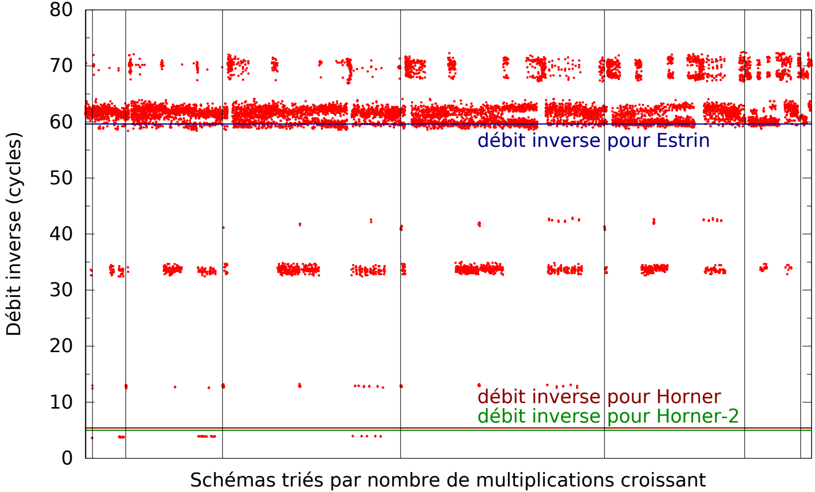
Comparisons with other schemes (2)

When x^{12} is a denormal number...



Comparisons with other schemes (2)

And when x^8 is a denormal number.



Outline

Impact on function approximation (1)

Implementation of a **faithful vectorized single-precision natural logarithm** for AVX2 architectures, using Intel Intrinsics Instructions:

- Logarithm approximation is quite simple
- Single-precision allows **native precision** computations for faithfulness
- Intrinsics enable **easy access to assembly instructions** and are **supported by GCC**.

Real world application: high-energy collision of two protons

- log was the **2$^{\text{nd}}$ most-called** function in a Cern's CMS simulation[2] ($>$ 7 M calls in $>$ 4300 traces)
- Average of 1600 calls/trace

[2] Piparo & Innocente 2016

Impact on function approximation (2)

Implementation details:

- Evaluates 4 **logarithms** at a time in 4 steps:
  1. Special treatment for denormals and inputs $> 2^{126}$
     - $x' = x \cdot 2^{23}$ or $x' = x \cdot 2^{-2}$
     - Then, $\ln(x) = \ln(x') + (-23||2) \cdot \ln 2$
  2. Range reduction using `RCPPS` : $u = x \cdot \circ\left(\frac{1}{x}\right) - 1$
  3. Table-based reduction, using `GATHER`: $\log_2\left(\circ\left(\frac{1}{x}\right)\right)$
  4. Polynomial evaluation and reconstruction:

$$\ln x \approx -\ln 2 \cdot \left(\log_2\left(\circ\left(\frac{1}{x}\right)\right) + (23||0) - (2||0)\right) + \ln(1+u)_{\text{poly}}$$

- **Degree-5 fp-minimax polynomial** beginning with $0 + x + \ldots$
- Used **Intrinsics backend** within CGPE, without FMA (FMA optimization handled by compiler)

Results (1) : Throughput and size of `logf_4`

(throughput in cycles/scalar result):

Compiler	Implementation	Interval	Throughput	Code + Data (B)
ICC 16.0.1	Intel SVML	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	**11** **4.5**	2200
GCC 6.1.0	GNU Libmvec	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	N/A **1.3**	**522**
	CGPE deg-5	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	43 7.6	1282
	Horner	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	44 **7.4**	**1266**
	Horner-2	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	42 7.5	1274
	Estrin	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	**41** **7.4**	1270

Caveats:

- Sizes for Intel SVML and GNU Libmvec only **for main routines**
- GNU Libmvec needs `-ffast-math` $\rightsquigarrow$ **no denormals, no exceptions**, . . .

Results (1) : Throughput and size of `logf_4`

(throughput in cycles/scalar result):

Compiler	Implementation	Interval	Throughput	Code + Data (B)
ICC 16.0.1	Intel SVML	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	**11** **4.5**	2200
GCC 6.1.0	GNU Libmvec	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	N/A **1.3**	**522**
	CGPE deg-5	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	43 7.6	1282
	Horner	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	44 **7.4**	**1266**
	Horner-2	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	42 7.5	1274
	Estrin	$[2^{-149}, 2^{-126}]$ $[2^{-12}, 2^{12}]$	**41** **7.4**	1270

Caveats:

- Sizes for Intel SVML and GNU Libmvec only **for main routines**
- GNU Libmvec needs `-ffast-math` $\rightsquigarrow$ **no denormals, no exceptions**, ...

Conclusions

- **Low impact** of the polynomial scheme at degree 5 for our implementation
- **High cost** of `mulps` for renormalization $\rightsquigarrow$ **integer ops** should be used instead

Results (2) : Accuracy of `logf_4`

Implementation	Interval	Max. error (ulps)	Incorrect rounding ratio
Intel SVML	$[2^{-149}, 2^{128} - 2^{104}]$	1	24%
GNU Libmvec	$[0.5, 1]$	2.25	67%
CGPE deg-5	$[2^{-149}, 2^{128} - 2^{104}]$	**0.997**	$\mathbf{2.8 \times 10^{-9}}$
Horner		**0.997**	$\mathbf{2.8 \times 10^{-9}}$
Horner-2		**0.997**	$\mathbf{2.8 \times 10^{-9}}$
Estrin		**0.997**	$\mathbf{2.8 \times 10^{-9}}$

Caveats:

- Intel SVML and GNU Libmvec use the **I/O precision** for intermediate computations

Results (2) : Accuracy of `logf_4`

Implementation	Interval	Max. error (ulps)	Incorrect rounding ratio
Intel SVML	$[2^{-149}, 2^{128} - 2^{104}]$	1	24%
GNU Libmvec	$[0.5, 1]$	2.25	67%
CGPE deg-5	$[2^{-149}, 2^{128} - 2^{104}]$	**0.997**	$\mathbf{2.8 \times 10^{-9}}$
Horner		**0.997**	$\mathbf{2.8 \times 10^{-9}}$
Horner-2		**0.997**	$\mathbf{2.8 \times 10^{-9}}$
Estrin		**0.997**	$\mathbf{2.8 \times 10^{-9}}$

Caveats:

- Intel SVML and GNU Libmvec use the **I/O precision** for intermediate computations

Conclusions:

- Our `logf_4` is **almost always correctly-rounded** (6 results are “only” faithful)

Outline

Conclusions

Automatic generation of polynomial evaluations should take into account

- **low-level, architecture-dependant details**: SIMD/SWAR, FMA available. . .
- the degree of the polynomial to be implemented
- if input domain and evaluation scheme may introduce **denormalization** (on specific architectures)
- the **compiler** (for vector code and FMA)

Perspectives

- Handle FMA in CGPE directly: $\neq$ latencies, maybe $\neq$ **low-latency schemes**
- **Automate the choice** of efficient schemes with respect to:
  - Floating-point arithmetic (binary32, binary64)
  - **(μ-)Architecture** (extensions, arithmetic units, registers, caches, . . .)
  - Polynomial degree
  - I/O precision/accuracy
  - Maybe more?
- Observe how **extended-precision algorithms** perform

Performances de schémas d'évaluation polynomiale sur architectures vectorielles

8ème Rencontres Arithmétique de l'Informatique Mathématique
Banyuls

Hugues de Lassus Saint-Geniès et Guillaume Revy

DALI, Université de Perpignan Via Domitia
LIRMM, Université de Montpellier
CNRS UMR 5506

28 – 30 juin 2016