January 19th 2015, HiPEAC15 - WAPCO, Amsterdam (NL)

Fast Exponential Computation on SIMD Architectures

A. Cristiano I. MALOSSI, Yves INEICHEN,
Costas BEKAS and Alessandro CURIONI
@ IBM Research – Zurich, Switzerland

Exa2Green
energy-aware numerics

SEVENTH FRAMEWORK PROGRAMME

Motivations

$$f(x) = e^x \equiv \exp(x),$$

Thousands of applications:

- Fourier transform
- Neural networks
- Lumped models
- Radioactive decay
- Population grown
- …

Existing techniques:

- Power series
- Look up tables
- IEEE-745 manipulation

Drawbacks of state of the art practices (1)

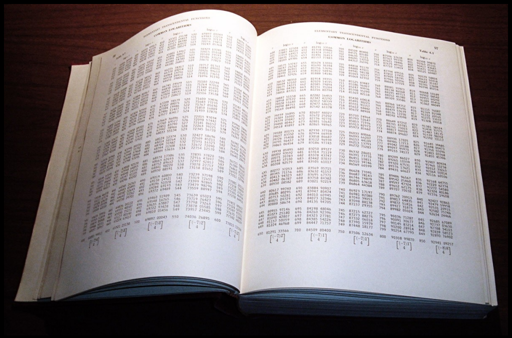
- Power series/Taylor expansions:

$$e^x = \sum_{k=1}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

PROS: make use of arithmetics (can use SIMD unit)
PROS: flexible accuracy
CONS: **convergence is very slow** (unusable for high accuracy)
CONS: even using Horner's rule it requires **too many floating-point multiply-add**

- Look-up tables:

$$e^x = 2^{x \log_2(e)} = 2^{x_i + x_f}, \quad \longrightarrow$$



PROS: faster than Power series/Taylor expansions
CONS: do not make use of arithmetics (**only partial SIMD use**)

Drawbacks of state of the art practices (2)

- **IEEE-745 manipulations, by N. N. Schraudolph in 1998:**

$$(-1)^s(1+m)\boxed{2^{x}}^{-x_0},$$

The idea is to profit from the power of 2 implied in the floating point representation

sxxxxxxx	xxxxmmmm	mmmmmmmm	mmmmmmmm	mmmmmmmm	mmmmmmmm	mmmmmmmm	mmmmmmmm
1	2	3	4	5	6	7	8
iiiiiiii	iiiiiiii	iiiiiiii	iiiiiiii	jjjjjjjj	jjjjjjjj	jjjjjjjj	jjjjjjjj

Figure 1: Bit representation of the `union` data structure used by the `EXP` macro. The same 8 bytes can be accessed either as an IEEE-754 `double` (top row) with sign `s`, exponent `x`, and mantissa `m`, or as two 4-byte integers `i` and `j` (bottom).

$$\texttt{int i} = A \cdot x + B - C, \qquad (4)$$

with $A = S/\ln(2)$, $B = S \cdot 1023$, $C = 60801$, being $S = 2^{20}$

PROS: extremely fast

CONS: very inaccurate (max 1 or 2 digits accurate)

Derivation of methodology (1)

IDEA: combine **IEEE-745 manipulations** with **polynomial interpolation**

Starting point:

$$\texttt{int i} = A \cdot x + B - C, \qquad (4)$$

with $A = S/\ln(2)$, $B = S \cdot 1023$, $C = 60801$, being $S = 2^{20}$

1) Use a single 64 bit "long int" to profit from 52 digits of mantissa (i.e., $S=2^{52}$);

2) Set $C=0$ (useless in our method)

3) Write the equality

$$e^x = 2^{x_i} \cdot 2^{x_f} \approx 2^{x_i} \cdot (1 + m - \mathcal{K}),$$

4) Determine the analytic correction factor

$$\mathcal{K} = 1 + m - 2^{x_f}.$$

5) note that $m \equiv x_f$, so

$$\mathcal{K}(x_f) = 1 + x_f - 2^{x_f},$$

Derivation of methodology (2)

IDEA: combine **IEEE-745 manipulations** with **polynomial interpolation**

6) We model the correction function $\mathcal{K}(x_f)$ with a polynomial $\mathcal{K}_n(x_f)$ in the form

$$\mathcal{K}_n(x_f) = a \cdot x_f^n + b \cdot x_f^{n-1} + c \cdot x_f^{n-2} + \ldots,$$

where $n \in [1, 10]$ denotes the order of the polynomial interpolation. The coefficients $\{a, b, c, \ldots\}$ are pre-computed according to the chosen interpolation.

7) Last, we plug our polynomial in the original expression, i.e.,

$$\texttt{long int i} = A\left(x - \ln(2) \cdot \mathcal{K}_n(x_f)\right) + B. \qquad (6)$$

Algorithm work-flow

Algorithm I   **Input:** x and n;   **Output:** $f(x) \approx e^x$

1: $x = x \cdot \log_2(e)$

2: $x_f = x - \texttt{floor}(x)$

3: $x = x - \mathcal{K}_n(x_f)$, with $\mathcal{K}_n(x_f) = a \cdot x_f^n + b \cdot x_f^{n-1} + c \cdot x_f^{n-2} + \ldots$

4: Compute the `long int i` as: $2^{52} \cdot x + \mathrm{B}$

5: Read the `long int i` as a `double` and return the value as the approximated exponential e^x

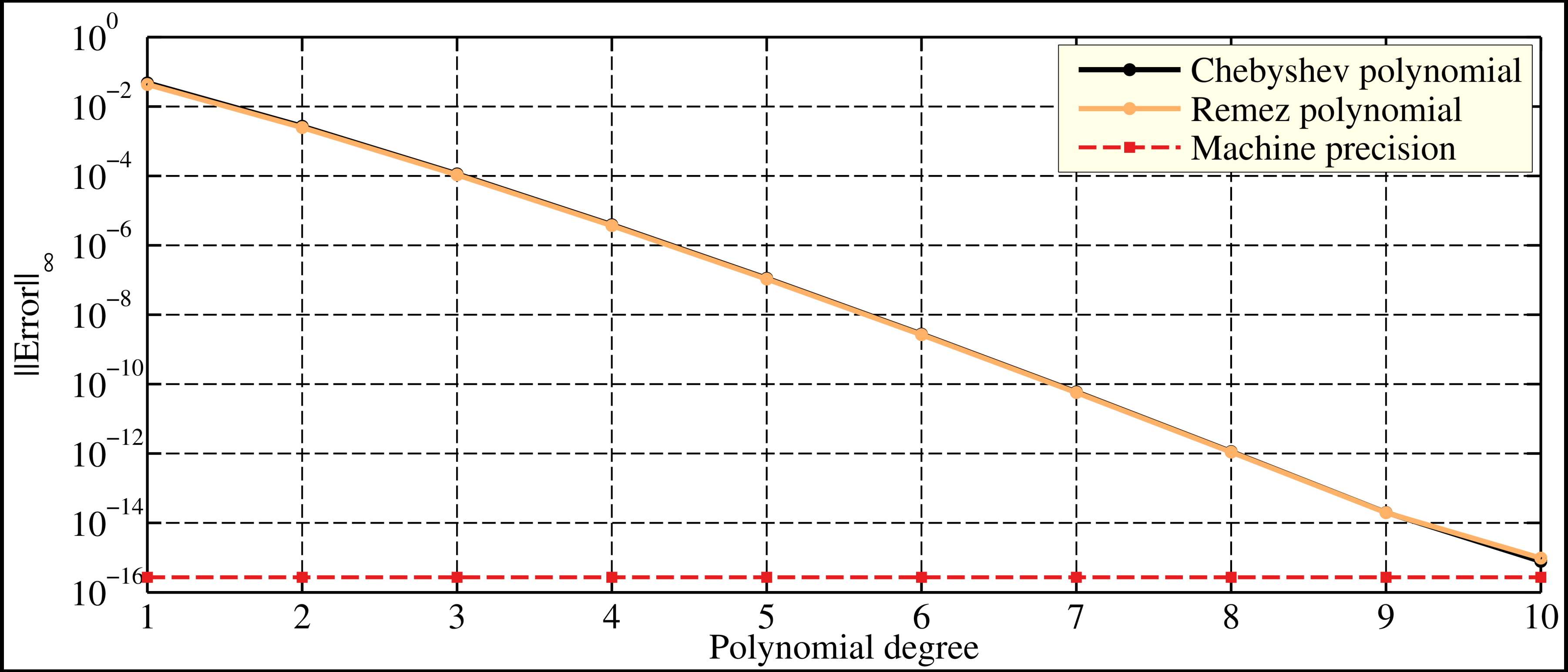
Main features

- All the computational instructions are on the **SIMD unit**
- Architecture independent implementation are possible for the scalar version (no SIMD)

Additional notes

- The constants 2^{52}, $\log_2(e)$, B as well as {a, b, c, …} are precomputed (no additional cost)
- Steps 3, 4, and 5 can be joined in a single code line (improve compiler optimization)
- In C++ step 5 can be performed as a reinterpret_cast<double &>

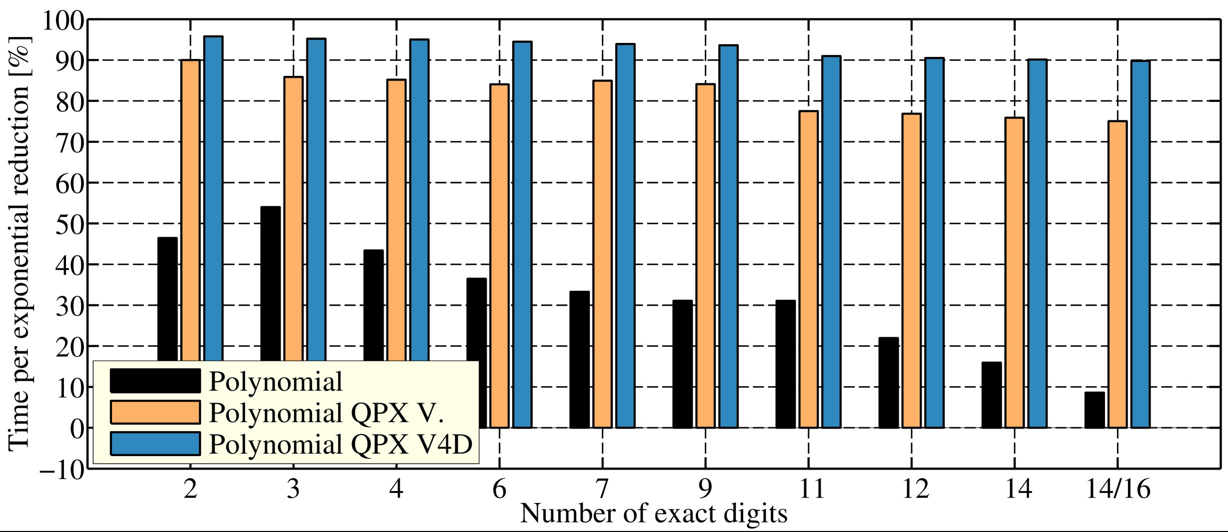
Flexible accuracy



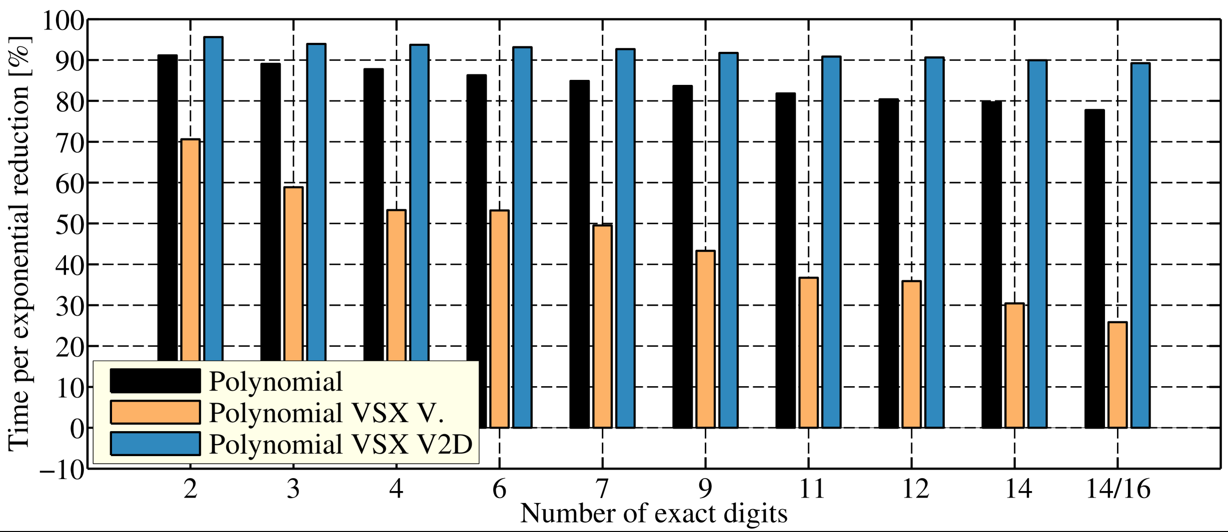
- Different polynomials can be employed in the same scheme (same cost)
- Accuracy can be tuned changing the degree of the polynomial
- Each degree implies an additional “FMADD”

Performance on L1-cache: time percent reduction

- **IBM BG/Q**

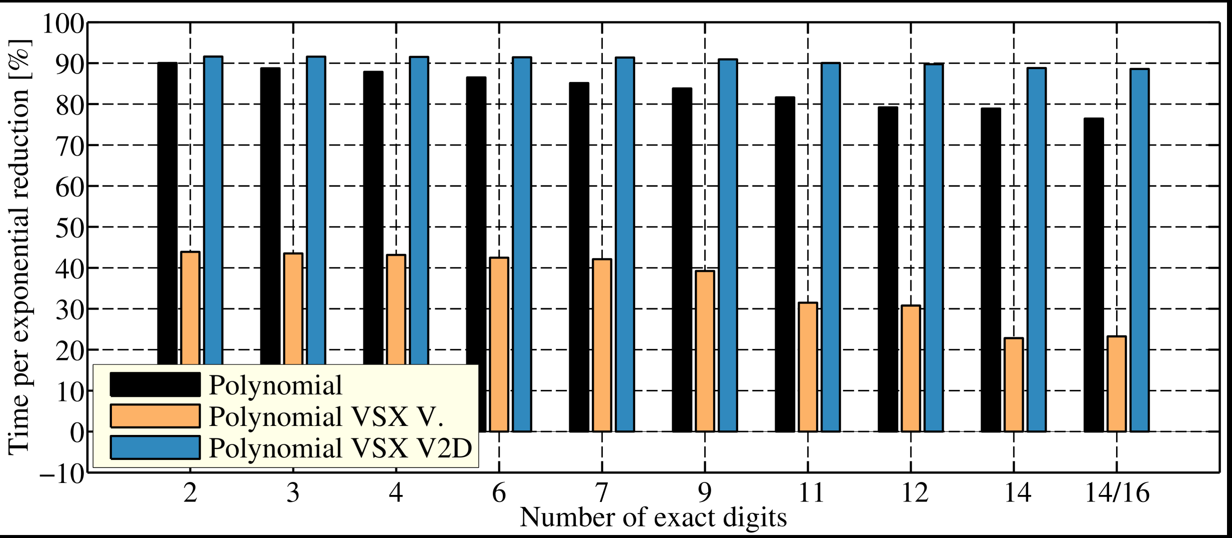


- **IBM Power 755**

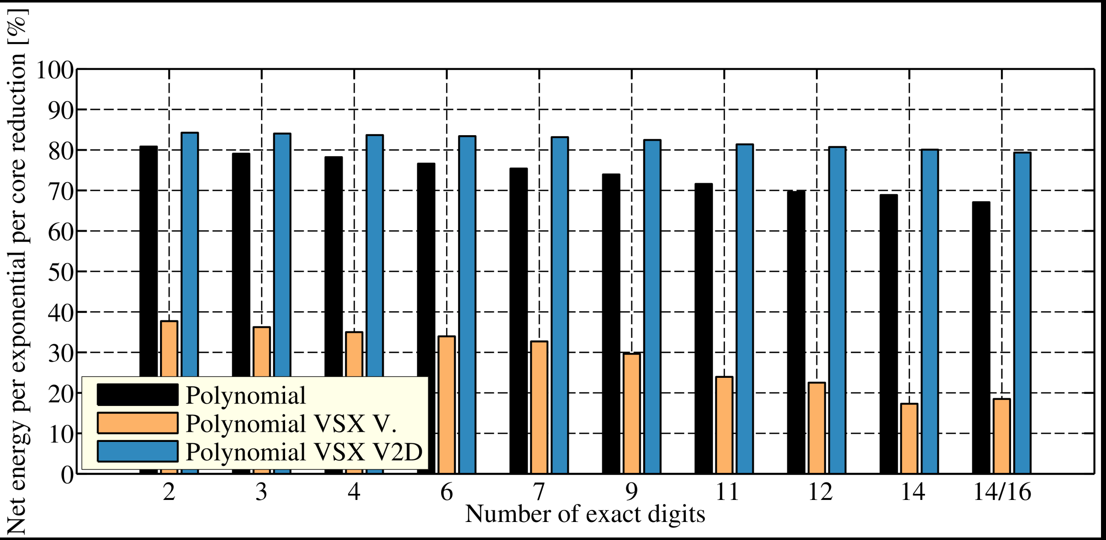


Performance on DDR RAM on Power 755 (similar results on BG/Q)

- **Time per exponential (percent reduction)**



- **Net energy per exponential (percent reduction)**



Summary and conclusions

Main ingredients

- IEEE-745 manipulation is **very fast**
- Polynomial interpolation is suitable for **100% SIMD unit use** (multiply-add instructions)

Main resulting advantages

- Huge **reduction in the time-to-solution** (up to 96 % on IBM BG/Q and Power7)
- Huge **reduction in the energy-to-solution** (up to 93 % on IBM BG/Q and Power7)
- Low-to-High **accuracy flexibility** (the user can control the degree of the polynomial)
- **Architecture flexibility** (specific SIMD implementation possible on any modern architecture)

Further advantages

- Scalar versions (no SIMD) are still much faster and energy efficient than classical strategies
- OpenMP (multithread) implementations possible and efficient for big vectors