Polynomial Evaluation on modern CPU architectures

T. Ewart[a], S. Yates[a], F. Delalondre[a], F. Schürmann[a]

[a] *Blue Brain Project, Campus Biotech, Ch. des Mines 9. CH-1212 Genève*

Abstract

In the evaluation of the elementary functions *e.g.* e^x, the challenge consists to compute with the maximum performance a polynomial of low degree polynomial (*e.g.* 10) in a small x-range, where the coefficients are known in advance. In this paper, we evaluate the main methods of the literature and we complete this work by the factorisation evaluation, that is surprisingly neglected in the literature. We focus our evaluation on the last modern processors of Intel(SandyBridge Haswell)/IBM(Power8)/ARM(V8) where several computational units are available, and may maximise the computing throughput. We show experimentally, for a polynomial of degree 10, the factorisation scheme combined with usual approach is the is the fastest.

1. State of the art

In evaluating the polynomial,

$$P(x) = a_0 + a_1 x + \ldots + a_n x^n, \ (a_i \in \mathbb{R}) \tag{1}$$

... Brute force ... Horner method ... k^{th}-order Horner ... Estrin ... adaptation coefficient .... factorisation ... challenge: maximum parallelism and minimise data hazard pipelining to fill up pipeline, avoid bubble, maximise performance

2. Algorithms

2.1. Evaluation of Powers

... Right-to-left binary method for exponentiation ... not the fastest ... but enough for us ... because did at compile time. $\lfloor log(n) \rfloor + \upsilon(n)$ multiplications where $\upsilon(n)$ number of ones in the binary representation of n

2.2. Brute force

... evaluate polynomial 1 at $x = x_0$ ... using evaluation of power as described previously n addition, $2n - 1$ multiplication ... degree parallelism n

2.3. Horner method: classical - 1^{st}-order

.... the most classical .... *"Horner's rule"* .... evaluate polynomial 1 at $x = x_0$

$$P(x_0) = a_0 + x_0(a_1 + x_0(a_2 + x_0(\ldots a_n))) \tag{2}$$

Horner rules n multiplications and n additions. Good but data hazard pipelining

2.4. Horner method: k^{th}-order

Generalisation of the previous section ... Introduce parallelism ...

$$\begin{aligned} P(x_0) &= Q_0(x_0^k) + Q_1(x_0^k) + \ldots + Q_{k-1}(x_0^k) \quad (3) \\ Q_{k-1}(x_0^k) &= x_0^{k-1}(a_{k-1} + x_0^k(a_{2k-1} + x_0^k(a_{3k-1} + x_0^k(\ldots a_{m\leq n})))) \quad (4) \end{aligned}$$

$n + k - 1$ multiplications and n additions, k degree of parallelism. Classical Horner method $k = 1$

2.5. Horner method: Estrin

Another generalization of Horner's rule:

$$c_i^{(0)} = a_i + x_0 a_{i+1} \tag{5}$$

$$c_i^n = c_i^{(n-1)} + x_0^{2^n} c_{i+2^n}^{n-1} \tag{6}$$

$$P(x_0) = c_0^n \tag{7}$$

look like

$$\begin{aligned} P(x_0) = \; & a_0 + a_1 x_0 \\ + \; & x_0^2(a_2 + a_3 x_0) \\ + \; & x_0^4(a_4 + a_5 x_0 + x_0^2(a_6 + a_7 x_0)) \\ + \; & x_0^8(a_8 + a_9 x_0 + x_0^2(a_{10} + a_{11} x_0) + x_0^4((a_{12} + a_{13} x_0 + x_0^2(a_{14} + a_{15} x_0))))) \\ & \ldots \end{aligned} \tag{8}$$

n multiplication, n additions, remark pattern design for FMA

2.6. factorization

factorize the polynomial (outside), get produce of linear or quadratic if complex-root. combinaison different method *e.g.* k factor gives $k - 1$ factor multiplication, factor depends of the method of multiplications.

2.7. Notations

P_m^n polynomial of degree n, m indicate the method (e = Estrin, h^k = Horner at the the k order, b = Brute Force) how we compute

2.8. *Combinatory*

Starting point Polynomial degree 10 approximation e^x no real root, complex only, 5 pair conjugate, 5 quadratic. How many factorization scheme? Partition of 10 into even summands (equivalent decomposition of 5 "multiply by 2"). $P^{10} = P^6P^4 = P^6P^2P^2 = P^4P^4P^2 = P^4P^2P^2P^2 = P^2P^2P^2P^2P^2$. Multiplication commutative. Every polynomial degree+1 method of evaluation (bruteforce + estrin + horner kth = degree-1) (horner kth=degree equivalent order one). if $P^n¶^m$ and $n! = m$ so $n \times m$ possibilitties. if $n = m$, $\binom{n+k-1}{k}$ possibilities (and not $\binom{n}{k}$ because repetitions are allowed). On the present example 231 possibilities. Automatic program than generate all the possibilities, get all the factorisation scheme

3. Processor & methods

3.1. *Intel SandyBridge/Haswell architecture*

blabla

3.2. *Latency, throughput*

Latency: number of cycle needed to execute a single instruction. Throughput: number of cycle needed to execute the same instruction. e.g. Haswell add double precision latency 3 cycle, throughput 1 cycle. mean 2 independents additions takes 4 cycles where 6 if they are independents. From the 231 possibilities difficult to determine what will be the fastest method due to the interaction latency throughput.

3.3. *measurement tool*

IACA measure latency/throughput of C++ code, kernel that is pinned by marks into the code.

3.4. *Programming model*

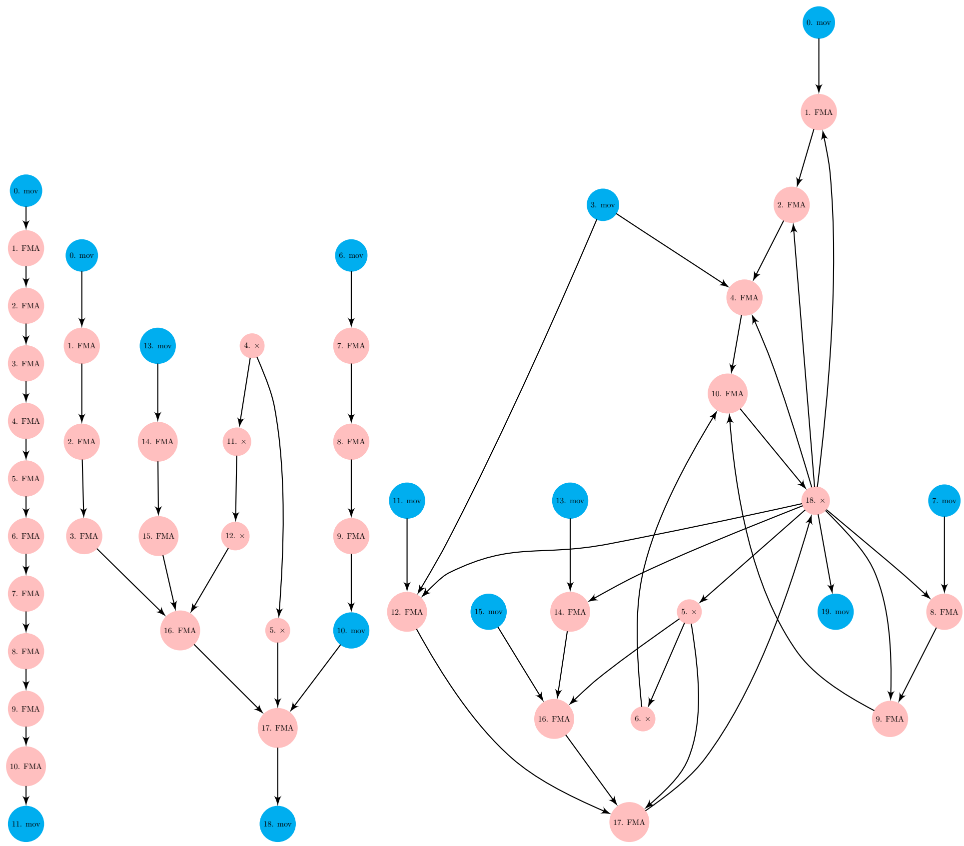
C++ basic recursive meta-programming (like template factorial), all models are written with fairplay. Assembly check, look good. Fine control of the code but compiler as the final word, and can do strong optimisation.

3.5. *ASM quality*

4. Results

5. Conclusions

factorisation is good



(a) (b) (c)

Figure 1: (a) classical Horner (57/5.5 [cycle]), (b) Estrin (32/7.1 [cycle]), (c) Estrin^6×BruteForce^4 (32/25 [cycle])

Table 1: Latency/Throughput add/mul/fma on the targeting architecture

	SandyBridge		Haswell		Power8		ARMv7	
add	3	1	3	0.8	6	1	5	2
mul	5	1	5	0.5	6	1	5	2
FMA	-	-	5	0.5	6	1	9	2
FPU/core	2		2		2		?	

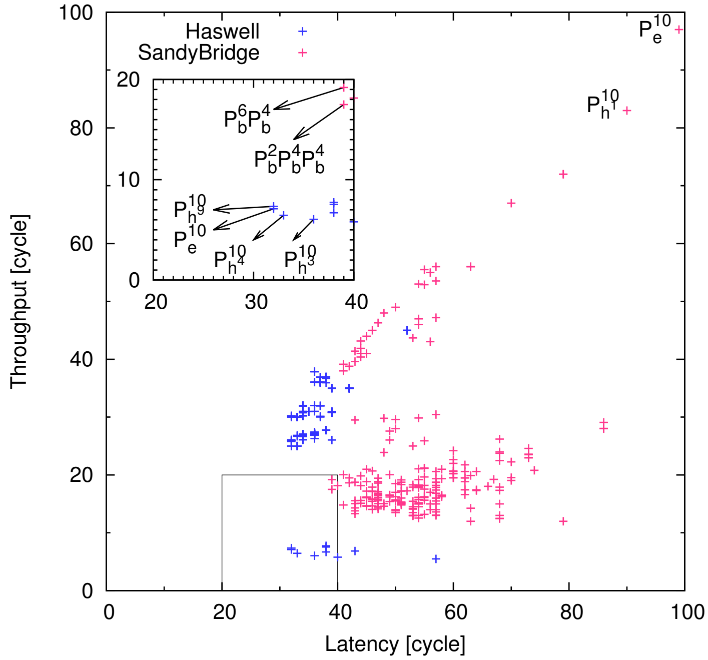


Figure 2: Latency/Throughput for SandyBridge & Haswell

Table 2: Latency [cycle] of the different algorithms for a polynomial of degree 10

	SandyBridge	Haswell	Power8	ARMv7
Brute force	54	38	56	60
Factorisation	44	39	32	31
Factorisation-Pan	44	39	32	32
Horner^{1st}	99	58	72	65
Horner^{2nd}	69	43	54	90
Horner^{3rd}	63	42	54	50
Horner^{4th}	60	39	54	50
Horner^{5th}	67	49	79	52
Horner^{6th}	65	44	79	67
Horner^{7th}	73	44	91	60
Horner^{8th}	74	44	81	74
Horner^{9th}	82	38	96	74
Horner^{10th}	79	38	92	74
Estrin	61	34	76	86