FAST SOFTMAX ON RV32IM
No FPU, No Problem

Software-Level Approximation of exp(x)

for AI Inference on RISC-V Embedded Cores

Yutung Cheng

Fast Softmax on RV32IM: Overview

- Problem
  - Softmax relies on exp( ), which is slow on RV32IM (without F)
- Proposed method
  - Taylor 3rd-order approximation with Horner's method
  - Optional LUT-based range decomposition (x = n + f)
- Impact
  - Accelerates softmax computation

Method	Time Reduction (512, fp32)	Max Error (input value [0, 10))
glibc::expf (basedline)	0%	0
Taylor 3rd (no LUT)	69%	0.0132
Taylor 3rd + LUT	58%	0.0003

Table 1.1, Experiment results

How glibc Calculate expf(x)

- Special Case Handling
  - Detect overflow (x > 88.72) and under flow (x < -103.97)
  - NaN, ±Inf, ± 0 all handled explicitly
- Decompose and range reduction (Eq.2.1, Eq.2.2)
  - Decompose exp(x) = 2^n * exp(r), n = round(x / ln(2))
  - Compute residual r = x - n * ln(2)
- LUT for 2^n
  - 2^n realized via precomputed 2^k table
- Polynomial Approximation for r
  - Use 3rd Taylor series on small r
- Resource Cost
  - Division for n, multiplication for r
  - Table lookup overhead
  - Float-to-int conversion

$$e^x = e^{\left(\frac{ln2}{N}K+\frac{ln2}{N}r\right)} = e^{\left(\frac{ln2}{N}K\right)}e^{\left(\frac{ln2}{N}r\right)}$$

Eq.2.1

$$e^{\left(\frac{ln2}{N}K\right)} = e^{\left(\frac{ln2}{N}(i*N+j)\right)} = 2^{i+\frac{j}{N}}$$

Eq.2.2

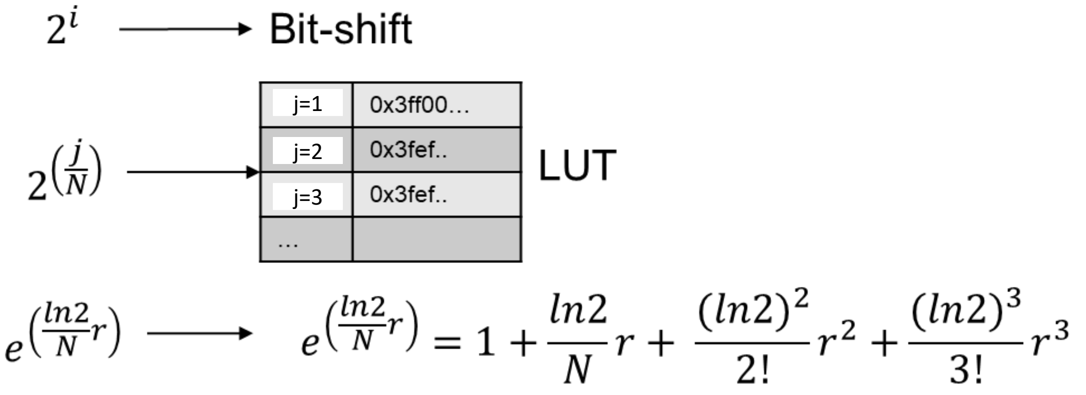

Fig.2.1, Overview of glibc strategy

How Cephes Calculate expf(x)

- Special Case Handling
  - Similar overflow / underflow guard as glibc
  - Early exit for x smaller than minimum, trun 0.0f
- Decompose and range reduction (Eq.3.1)
  - Decompose exp(x) = 2^n * exp(r)
  - n = floor(x / ln(2))
- LUT for 2^n
  - Bit shift to handle 2^n
- Polynomial Approximation for r (Eq.3.2)
  - Use Pade approximation, better accuracy in larger range
- Resource Cost
  - Avoid LUT, which helps memory
  - Pade approximation use division and rational

$$e^x = e^{f+kln(2)} = e^f \times 2^k$$

$$k = floor\left(\frac{x}{ln(2)} + 0.5\right)$$

Eq.3.1

$$e^f \approx \frac{P(f)}{Q(f)} = \frac{120 + 60f + 12f^2 + f^3}{120 - 60f + 12f^2 - f^3}$$

Eq.3.2

$$2^k \longrightarrow \text{Bit-shift}$$

$$e^f \longrightarrow e^f = \frac{P(f)}{Q(f)}$$

Fig.3.2, Overview of Cephes strategy

Proposed Strategy

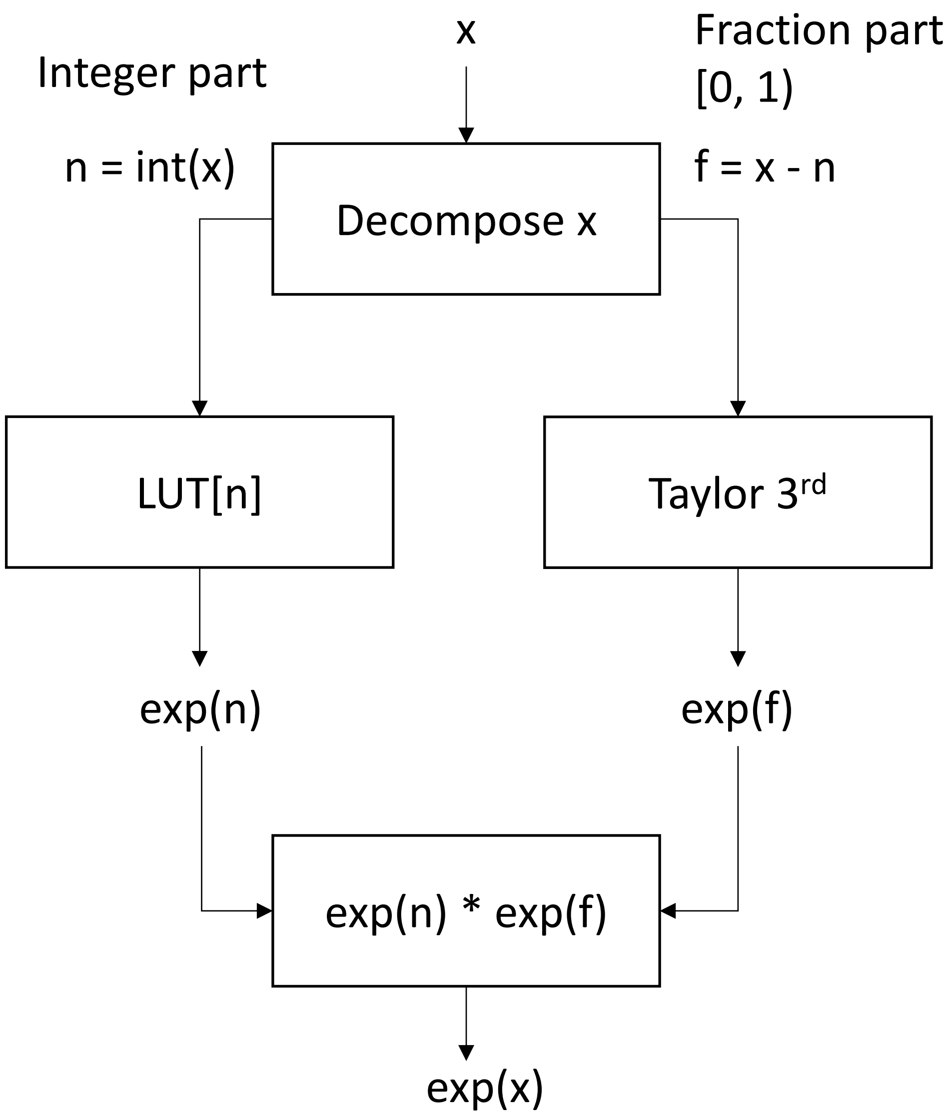

Fig.4.1, Overview of proposed strategy

- Input Decomposition
  - x = n + f, where n = int(x) and f = x - n
- LUT
  - Precomputed exp(n) for integer part in target range
  - Use n as index
- Taylor Approximation
  - 3rd degree polynomial over f ∈ [0,1)
- Horner's Method
  - Reduce multiplication overhead of Taylor approximation (Eq.4.1)

$$e^f \approx \big((T_3 f + T_2)f + T_1\big)f + 1$$

Eq.4.1

Conclusion and Future Work

- Problem
  - Costly exp(x) on RV32IM without FPU
- Solution
  - Decompose input to int and fraction
  - Use LUT for int part
  - Use Taylor-3 approx. for fraction
- Benefit
  - Reduce computation cycles
- Future work
  - Explore log-sum-exp softmax
  - Deploy on RV32IM with HW counter

Method	Time Reduction (512, fp32)	Max Error (input value [0, 10))
glibc::expf (basedline)	0%	0
Taylor 3rd (no LUT)	69%	0.0132
Taylor 3rd + LUT	58%	0.0003

Table 1.1, Experiment results