Cybozu Labs

expの実装に見る AVX-512とSVEの違い

2020/12/3

第9回HPC-Phys勉強会

光成滋生

目次

Cybozu Labs

- expの近似計算
  - AVX-512による実装
  - SVEによる実装
  - レジスタリネーミング
- SVEのfexpaを使った近似計算
  - 実装
- 逆数近似命令

exp(float x);の近似計算（1/3）

CybozuLabs

- 級数展開を使う
  - $e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots$
- 要請
  - 級数展開におけるxは小さいのが望ましい
  - 2^n（nが整数）は高速に求められる
- 式変形
  - $e^x = 2^{x\log_2(e)}$
  - $y \coloneqq x\log_2(e)$を整数部分nと小数部分aに分ける（$|a| \leq 1/2$）
    - $y = n + a$
  - $e^x = 2^y = 2^{n+a} = 2^n 2^a$
    - $2^a = e^{a\log(2)}$
    - $b \coloneqq a\log(2),\ |b| \leq \log(2)/2 = 0.346$

exp(float x);の近似計算（2/3）

Cybozu Labs

- floatなら1e-6程度まで求まれば十分
  - 5次の項まで計算
  - $e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120}$
- 係数補正
  - $f(x) \coloneqq 1 + x + Bx + Cx^2 + Dx^3 + Ex^4 + Fx^5$
  - $x \in [-L, L]$ where $L \coloneqq \log(2)/2$
  - $I \coloneqq \int_{-L}^{L} (f(x) - e^x)^2 dx$を最小化する$(B, C, D, E, F)$を求める
    - 1次の項(A)を1に固定しているのは0次と共用したいから
    - 誤差が概ね半分程度になる
    - Sollyaのremezよりは誤差が小さい?
      - L2ノルムとL1ノルムどちらがよいかアプリ依存?

exp(float x);の近似計算（3/3）

Cybozu Labs

- まとめ

```
input : x
x = x * log_2(e)
n = round(x) ; 四捨五入
a = x - n
b = a * log(2)
c = 1 + b(1 + b(B + b(C + b(D + E b))))
y = c * 2^n
output : y
```

- 配列に対する一括処理

```
void expv(float *dst, const float *src, size_t n) {
  for (size_t i = 0; i < n; i++) dst[i] = exp(src[i]);
}
```

  - テーブルを使わないためSIMD化しやすい

AVX-512

- Intelの512bit幅のSIMD命令セット
  - 32個の512bit AVXレジスタ
  - double x 8, float x 16, int32 x 16, int16 x 32, int8 x 64などの型
  - 概ね3オペランド
    - op(r1, r2, r3) ; r1 = r2 op r3

```
paddd zmm0, zmm1, zmm2 // zmm0 = zmm1 + zmm2 as 32-bit int
addps zmm0, zmm1, zmm2 // as float
addpd zmm0, zmm1, zmm2 // as double
```

      - d ; 32-bit int, ps ; float, pd ; double
    - 積和演算FMA
      - d = a * b + cが望ましいが4オペランドになるので

```
vfmadd132ps r1, r2, r3 // r1 = r1 * r3 + r2
vfmadd213ps r1, r2, r3 // r1 = r2 * r1 + r3
vfmadd231ps r1, r2, r3 // r1 = r2 * r3 + r1
```

Xbyak/Xbyak_aarch64

- C++でCPUのニーモニックを記述
- 実行時にそれに対応する機械語が生成されて実行する
- 例：「整数nを足す関数」を生成する関数

```
struct Code : Xbyak::CodeGenerator {
  void genAdd(int n) {
    // rsiはLinuxでの関数の第一引数
    lea(rax, ptr[rsi + n]);
    ret();
  } };
```

- genAdd(5); // 「5を足す関数」が実行時に生成される
  - 生成された関数

```
lea rax, [rsi + 5] // + 5は即値
ret
```

  - exp自体は静的な関数でよいがIntel oneDNNではパーツの一つとして実行時生成されている

AVX-512によるexp

- round関数
  - vrndscaleps(dst, src, round_ctl)
    - round_ctl ; 2bitフラグ
      - 00 ; nearest even ; 一番近い偶数丸め
      - 01 ; equal or smaller ; 切り捨て
      - 10 ; equal or larger ; 切り上げ
      - 11 ; truncate ; 0方向に切り捨て
  - vrndscaleps(dst, src, 0) ; dst = round(src);
  - `c = 1 + b(1 + b(B + b(C + b(D + E b))))`
    - FMAを5回適用
  - $y = c \times 2^n$の部分
    - AVX2までは整数nをビットシフトしてfloatに変換して…
    - AVX-512ではvscalefps(dst, r1, r2) // dst = r1 * 2^r2

exp一つ分

- 準備
  - 下記変数はzmmレジスタのエリアス名
  - log2_e, log2, one, B, C, D, Eは事前に定数設定
  - t1, t2は一時レジスタ, xが入出力（x = exp(x)）

```
genExpOne() {
  vmulps(x, log2_e);          // x *= log2_e
  vrndscaleps(t1, x, 0);      // t1 = round(x)
  vsubps(x, t1);              // x = x - t1 ; 小数部分a
  vmulps(x, log2);            // a * log2
  vmovaps(t2, E);
  vfmadd213ps(t2, x, D);
  vfmadd213ps(t2, x, C);
  vfmadd213ps(t2, x, B);
  vfmadd213ps(t2, x, one);
  vfmadd213ps(t2, x, one); // t2 = 1+b(1+b(B+b(C+b(D+E b))))
  vscalefps(x, t2, t1);       // x = t2 * 2^t1
}
```

端数処理(1/2)

CybozuLabs

- ループnが16の倍数でないときの扱い
- マスクレジスタk1, ..., k7
  - zmmレジスタのkに対応するbitが1なら処理, 0なら非処理
  - T_zを指定すると非処理の部分に0が入る
- vmovups(zm0|k1|T_z, ptr[src]);

```
k1 = 0b00..01111111 (8bitの1)のとき
src |  0|  1|  2|  3|  4|  5|  6|  7|  8|  9|  a|  b| ...
                                    |← readしない        →
zm0 | x0| x1| x2| x3| x4| x5| x6| x7|  0|   |  0|  0|...
```

  - readしない部分はメモリアクセスしない(アクセス違反しない)
  - マスクレジスタを指定しないときに比べてやや遅い
    - 16の倍数のときは指定しない方がよい

端数処理(2/2)

Cybozu Labs

- コア部分

```
Label lp = L();                    // メインループ
    vmovups(zm0, ptr[src]);        // zm0 = *src
    add(src, 64);                  // src += 64
    genExpOne()                    // zm0 = exp(zm0)
    vmovups(ptr[dst], zm0);        // *dst = zm0
    add(dst, 64);                  // dst += 64
    sub(n, 16);                    // n -= 16
    jnz(lp);                       // if (n != 0) goto lp
L(mod16);                          // 端数処理
    and_(ecx, 15);                 // n &= 15
    jz(exit);
    mov(eax, 1);
    shl(eax, cl);                  // eax = 1 << n
    sub(eax, 1);                   // eax = (1 << n) - 1
    kmovd(k1, eax);                // k1 = nビットの1
    vmovups(zm0|k1|T_z, ptr[src]); // 残り読み
    genExpOne()                           // zm0 = exp(zm0)
    vmovups(ptr[dst]|k1, zm0|k1);  // 残り書き
L(exit);
```

ベンチマーク

CybozuLabs

- float x[16384];に対するstd::exp(float)との比較(clk)

std::exp	fmath::exp
140.1	8.7

  - Xeon Platinum 8280 2.7GHz, g++ 9.3.0 Ubuntu 20.04.1 LTS
- genOneExpをループアンロール

```
genExpTwo() {
  vmulps(x0, log2_e);
    vmulps(x3, log2_e);
  vrndscaleps(t1, x0, 0);
    vrndscaleps(t4, x1, 0);
  vsubps(x0, t1);
    vsubps(x3, t4);
  ...
```

- fmath::exp 8.7clk→7.2clk
  - https://github.com/herumi/fmath/blob/master/fmath2.hpp

SVE

Cybozu Labs

- Armの可変長SIMD命令セット
  - 128〜1024(?)bitレジスタ
  - double x 8, float x 16, int32 x 16, int16 x 32, int8 x 64などの型
  - 概ね3オペランド
  - A64FX（富岳のCPU）では32個の512bitレジスタz0, ..., z31

```
fadd z0.s, z1.s, z2.s // as float
add z0.b, z1.b, z2.b // as int8
```

  - movprfx（dstをsrcとして利用）

```
movprfx(dst, pred, r1);
fmadd(dst, pred, r2, r3); // dst = r1 * r2 + r3
```

    - predは述語レジスタ
      - マスクレジスタ相当
      - AVX-512と異なり常に指定

述語レジスタ

Cybozu Labs

- SVEレジスタの各要素を処理する(1)か否(0)かを指定
- 例 ld1w(z.s, p/T_z, ptr(src));
  - z = *src; // float x 16個読み込み

src	x0	x1	x2	x3...
z.s	x0	0	x2	x3...
p	1	0	1	1

- i番目の各要素(i = 0, ..., 15)について
  - 述語レジスタp[i] = 1ならz[i] = src1[i]
  - p[i] = 0ならT_z(zero)を指定しているのでz[i] = 0
  - T_zを指定しなければp[i]の値を変更しない

whilelt(p.s, x4, n);

CybozuLabs

- 「x4 + i < n」が成り立つ添え字までp[i] = 1にする
- 例 x4 + 16 <= nならi = 0, ..., 15についてp[i] = 1
  - 全てのデータが有効

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

- x4 + 3 = nならi ≦ 2についてp[i] = 1, その他p[i] = 0

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

  - p[i] = 0の部分はデータを読まない・書かない
    - 読み書き属性が無い領域でも大丈夫

ループの終わり処理

Cybozu Labs

- メイン部分

```
Label lp = L();
  ld1w(z0.s, p0/T_z, ptr(src1, x4, LSL, 2));// z0 = src1[x4 << 2]
 ...
  incd(x4);                              // x4 += 16
L(cond);
  whilelt(p0.s, x4, n); // while (x4 < n)なら
  b_first(lp);          // lpラベルにジャンプ
```

  - x4 + 16 ≦ nである限りp0[i] = 1 for i = 0, ..., 15
  - ループの最終ではp0[i] = 1 for i <(n % 16), p0[i] = 0(otherwise)
- メリット
  - SVEが256bitや1024bitでも同じコードで動く
  - SVEはScalable Vector Extensionの略
- デメリット
  - あまり速くない;結局AVX-512のように分けた方がよい

sveによるgenExpOne

CybozuLabs

- AVX-512とほぼ同じ

```
fmul(tz0, tz0, log2_e);
frintn(tz2, p, tz0); // round : float -> float
fcvtzs(tz1, p, tz2); // float -> int
fsub(tz2, tz0, tz2);
fmul(tz2, tz2, log2);
movprfx(tz0, p, E);
fmad(tz0, p, tz2, D); // tz0 = tz2 * E + D
fmad(tz0, p, tz2, C);
fmad(tz0, p, tz2, B);
fmad(tz0, p, tz2, one);
fmad(tz0, p, tz2, one);
fscale(tz0, p, tz1);        // tz0 *= 2^tz1
```

- 述語レジスタpはall 1に設定
- round関数はfrintn
- fscale(x, p, n)のnがfloatではなくintなのでfcvtzsでintに変換

レジスタリネーミングが苦手?

Cybozu Labs

- frintnなどの命令のあとの依存関係
  - レジスタ名を入れ換えると2倍速くなるケース(74→32nsec)
  - https://github.com/herumi/misc/commit/0362d5647f693be66da841eea7ca333d0f5b5329

```
125  125            fmul(z0.s, z0.s, log2_e.s);
126  126            frintn(z2.s, p, z0.s); // rounding : float -> float
127  127            fcvtzs(z1.s, p, z2.s); // float -> int
128       -         fsub(z0.s, z0.s, z2.s);
129       -         fmul(z0.s, z0.s, log2.s);
130       -         movprfx(z2.s, p, expCoeff[4].s);
131       -         fmad(z2.s, p, z0.s, expCoeff[3].s);
132       -         fmad(z2.s, p, z0.s, expCoeff[2].s);
133       -         fmad(z2.s, p, z0.s, expCoeff[1].s);
134       -         fmad(z2.s, p, z0.s, expCoeff[0].s);
135       -         fmad(z2.s, p, z0.s, expCoeff[0].s);
136       -         movprfx(z0.s, p, z2.s);
137       -         fscale(z0.s, p, z1.s); // z0 = z2 * 2^z1
     128  +         fsub(z2.s, z0.s, z2.s);
     129  +         fmul(z2.s, z2.s, log2.s);
     130  +         movprfx(z0.s, p, expCoeff[4].s);
     131  +         fmad(z0.s, p, z2.s, expCoeff[3].s);
     132  +         fmad(z0.s, p, z2.s, expCoeff[2].s);
     133  +         fmad(z0.s, p, z2.s, expCoeff[1].s);
     134  +         fmad(z0.s, p, z2.s, expCoeff[0].s);
     135  +         fmad(z0.s, p, z2.s, expCoeff[0].s);
     136  +         fscale(z0.s, p, z1.s); // z0 *= 2^z1
138  137       }
```

ループアンロール

Cybozu Labs

- 利用レジスタを増やしてループ展開

```
// n = 1 or 2 or 3
for (int i = 0;i<n;i++) fmul(t[0+i*3], t[0+i*3], log2_e);
for (int i = 0;i<n;i++) frintn(t[2+i*3], p, t[0+i*3]);
for (int i = 0;i<n;i++) fcvtzs(t[1+i*3], p, t[2+i*3]);
for (int i = 0;i<n;i++) fsub(t[2+i*3], t[0+i*3], t[2+i*3]);
for (int i = 0;i<n;i++) fmul(t[2+i*3], t[2+i*3], log2);
for (int i = 0;i<n;i++) {
  movprfx(t[0+i*3], p, E);
  fmad(t[0+i*3], p, t[2+i*3], D);
}
for (int i = 0;i<n;i++) fmad(t[0+i*3], p, t[2+i*3], C);
for (int i = 0;i<n;i++) fmad(t[0+i*3], p, t[2+i*3], B);
for (int i = 0;i<n;i++) fmad(t[0+i*3], p, t[2+i*3], one);
for (int i = 0;i<n;i++) fmad(t[0+i*3], p, t[2+i*3], one);
for (int i = 0;i<n;i++) fscale(t[0+i*3], p, t[1+i*3]);
```

ベンチマーク

- float x[16384];に対するstd::exp(float)との比較(nsec)
  - FX700

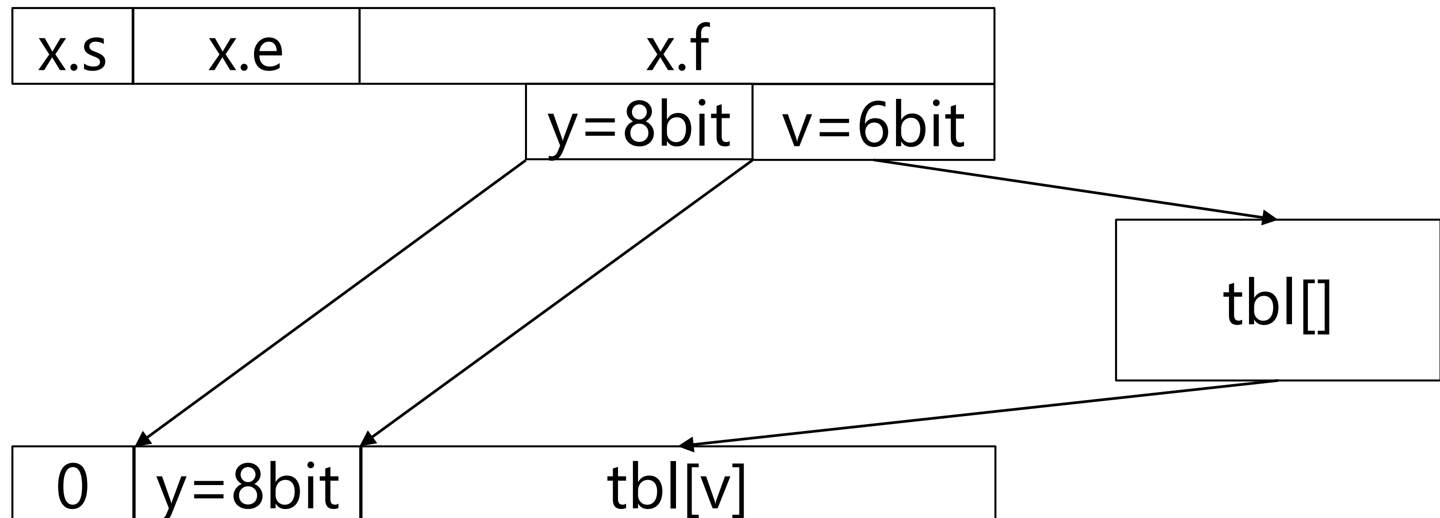
std::exp	fmath::exp, N=1	N=2	N=3
217.3	19.0	11.8	8.4

  - A64FXはXeonに比べてレイテンシが大きい
    - https://github.com/fujitsu/A64FX
      - frintn, fmul, fadd; 9clk
      - fdiv, 98clk
    - ループアンロールの効果が大きい

SVEのfexpa

CybozuLabs

- pow(2, i/64) (i=0,...,63)のテーブル引き命令
- floatのbit表現
  - u=|s:1|e:8|f:23|, $f = (-1)^s 2^{e-127}(1 + \frac{f}{2^{24}})$, e:指数部, f:仮数部
  - tbl[i] := 「pow(2, i/64)」の下位23bit
    - $x = 2^{\frac{i}{64}}$ for $i = 0, \ldots, 63$は$1 \le x < 2$なので常にx.e=127
- fexp(x)の挙動



fexpaの使い方

Cybozu Labs

- 2ベキの近似なのでそれにもっていく
- $\exp(x) = 2^{x\log_2(e)} = 2^y$ where $y := x\log_2(e)$
- $y = n + a$ where $n :=$ floor(y), $a := n - y$, $0 \le a < 1$
- fexpaの添え字vが2ベキになるには[1, 2)にある必要性
- $b := 1 + a$とすると$1 \le b < 2$, $y = (n-1) + b$

b.f	
v=6bit	w=17bit

  - b.f上位6bit vを取り出して使う c = fexpa(b.f >> 17)
  - $b = b.\mathrm{f}/2^{24}$=v /2^6 + w/2^{24}, $z :=$ w/2^{24}
  - 2^b =fexpa(v)$\cdot$ 2^z, $2^z = e^{\log(2)z} = 1 + \log(2)\,z + (\log(2)\,z)^2/2$
  - 差分が小さいので2次の項まででよい
  - https://github.com/herumi/misc/blob/master/sve/fmath-sve.hpp

fexpaによるexp

CybozuLabs

- 主要コード概要

```
fmul(t0, t0, para.log2_e);
frintm(t1, p, t0);          // floor : float -> float
fcvtzs(t2, p, t1);          // n = float -> int
fsub(t1, t0, t1);           // a
fadd(t1, t1, one);          // b = 1 + a
lsr(t3, t1, 17);            // v = b >> 17
fexpa(t3, t3);              // c = fexpa(v)
fscale(t3, p, t2);          // t3 *= 2^n
and_(t2.d, t1.d, not_mask17.d);
fsub(t2, t1, t2);           // z
movprfx(t0, p, coeff2);
fmad(t0, p, t2, coeff1);
fmad(t0, p, t2, one);       // 1 + log2 z + (log2)^2/2 z^2
fmul(t0, t3, t0);
```

ベンチマーク

Cybozu Labs

- float x[16384];に対するstd::exp(float)との比較(nsec)
  - ()内はfxpaを使わないバージョン

std::exp	N=1	N=2	N=3
217.3	18.2 (19.0)	11.3 (11.8)	8.4 (8.4)

  - 気持ち速いかも?
  - 定数レジスタの個数 7 vs 5
  - 中間レジスタの個数 3 vs 4
- 雑感
  - 最初の方法に比べてレジスタの依存関係が複雑になる
    - 定数レジスタが少ないのはよいがそこまでのメリットが
    - もう少し精度があれば
    - うまく速くなるレジスタ割り当てに試行錯誤（全数探索?）

逆数近似計算

Cybozu Labs

- fdivの代わりに逆数近似命令(rcp)のあと補正する

```
t = rcp(x)
1/x = 2 * t - x t^2
```

- AVX-512

```
// x : input/output
// t : temporary
// two : 2.0f
vrcp14ps(t, x);
vfnmadd213ps(x, t, two); // x = -xt + 2
vmulps(x, x, t);
```

- SVE ; frecps(dst, src1, src2) ; // dst = 2 - src1 * src2
  - 2回補正するとfloatに近い精度

逆数近似計算

CybozuLabs

- frintmあるいはaddの後のz0の逆数処理
  - https://github.com/herumi/misc/blob/master/sve/inv.cpp
    - ○で囲った部分が変?
    - レジスタリネーミング?

// input : z0 (compute it with z0, z1, z2)	clk frintm	add
1. fdivr(z0.s, p0, one.s);	100	100
2. frecpe(z1.s, z0.s); frecps(z2.s, z0.s, z1.s); fmul(z0.s, z1.s, z2.s);	33	7.9
3. frecpe(z1.s, z0.s); frecps(z3.s, z0.s, z1.s); fmul(z0.s, z1.s, z3.s);	10.9	7.9

4. frecpe(z1.s, z0.s); frecps(z2.s, z0.s, z1.s);
fmul(z1.s, z1.s, z2.s);
frecps(z2.s, z0.s, z1.s); fmul(z0.s, z1.s, z2.s);
	51	11.0

5. frecpe(z1.s, z0.s); frecps(z3.s, z0.s, z1.s);
fmul(z1.s, z1.s, z3.s);
frecps(z3.s, z0.s, z1.s); fmul(z0.s, z1.s, z3.s);
	11.2	11.0

まとめ

- AVX-512とSVE(A64FX)
  - どちらも512-bitレジスタx32
  - SVEの方がレイテンシが大きい
    - ループアンロール重要
    - レジスタリネーミングに気をつける場合がある?