VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Semestrálny projekt – IPA 2019/2020

Adam Múdry (xmudry01) 22. decembra 2019

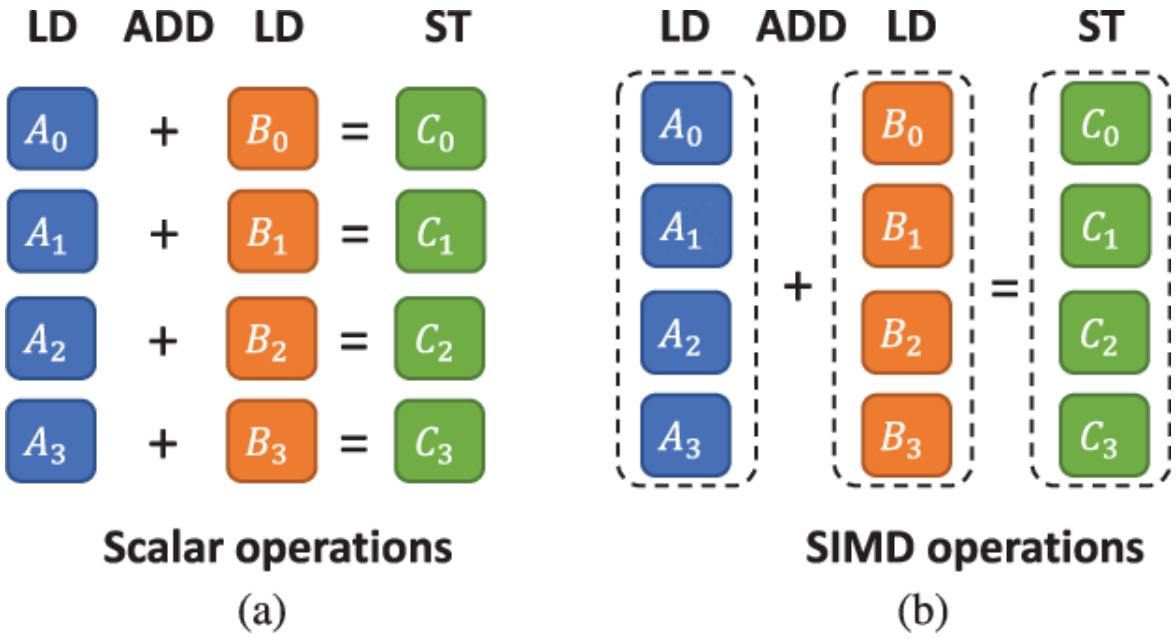
1 Úvod

Úlohou bolo optimalizovať a implementovať určité funkcie / algoritmy v programe na trénovanie doprednej neurónovej siete pomocou asembleru a SSE / AVX / AVX2 inštrukcií.

Modifikované súbory: Matrix.cpp, Network.cpp

2 Návrh riešenia

Moje riešenie pozostáva z využitia SSE inštrukcií na urýchlenie výpočtov sčítania, odčítania a násobenia matíc s číslami typu float alebo matíc s maticami. Ďalej som chcel podobným spôsobom riešiť aj funkciu *transpose()* (nestihol som prerobiť správne indexovanie, chcel som využiť intrinsic funkciu *_MM_TRANSPOSE4_PS* a transponovať matice po blokoch) a *sigmoid()*, avšak to sa mi nepodarilo a kvôli časovému sklzu tú optimalizované iba funkcie *add()*, *sub()* a *mul()*.



Obr. 1: Vizualizácia funkovanie vektorových SSE inštrukcií [1]

3 Implementácia

Riešenie som implementoval pomocou intrinsic funkcií *_mm_load_ps()*, *_mm_store_ps()*, *_mm_add_ps()*, *_mm_sub_ps()* a *_mm_mul_ps()*, ktoré využívajú inštrukčnú sadu SSE a jej rozšírené registre. Nasleduje ukážka optimalizácie funkcie pre násobenie.
Pôvodné riešenie:

```
for (i=0 ; i<height ; i++) {
    for (j=0 ; j<width ; j++) {
        result.array[i][j] = array[i][j] * m.array[i][j];
    }
}
```

Moje riešenie:

```
__m128 array4_1_0, array4_2_0, array4_1_1, array4_2_1,
array4_1_2, array4_2_2, array4_1_3, array4_2_3;
i = 0;
for (; i + 3 < height; i+=4) {
    j = 0;
    for (; j + 3 < width; j+=4) {

        array4_1_0 = _mm_load_ps((float*)&array[i][j]);
        array4_2_0 = _mm_load_ps((float*)&m.array[i][j]);

        array4_1_1 = _mm_load_ps((float*)&array[i+1][j]);
        array4_2_1 = _mm_load_ps((float*)&m.array[i+1][j]);

        array4_1_2 = _mm_load_ps((float*)&array[i+2][j]);
        array4_2_2 = _mm_load_ps((float*)&m.array[i+2][j]);

        array4_1_3 = _mm_load_ps((float*)&array[i+3][j]);
        array4_2_3 = _mm_load_ps((float*)&m.array[i+3][j]);

        array4_1_0 = _mm_mul_ps(array4_1_0, array4_2_0);
        array4_1_1 = _mm_mul_ps(array4_1_1, array4_2_1);
        array4_1_2 = _mm_mul_ps(array4_1_2, array4_2_2);
        array4_1_3 = _mm_mul_ps(array4_1_3, array4_2_3);

        _mm_store_ps((float*)&result.array[i][j], array4_1_0);
        _mm_store_ps((float*)&result.array[i+1][j], array4_1_1);
        _mm_store_ps((float*)&result.array[i+2][j], array4_1_2);
        _mm_store_ps((float*)&result.array[i+3][j], array4_1_3);
    }
    for (; j < width ; ++j) {
        result.array[i][j]   = array[i][j]   * m.array[i][j];
        result.array[i+1][j] = array[i+1][j] * m.array[i+1][j];
        result.array[i+2][j] = array[i+2][j] * m.array[i+2][j];
        result.array[i+3][j] = array[i+3][j] * m.array[i+3][j];
    }
}
for (; i < height; ++i) {
    j = 0;
    for (; j + 3 < width; j+=4) {
        array4_1_0 = _mm_load_ps((float*)&array[i][j]);
        array4_2_0 = _mm_load_ps((float*)&m.array[i][j]);
        array4_1_0 = _mm_mul_ps(array4_1_0, array4_2_0);
        _mm_store_ps((float*)&result.array[i][j], array4_1_0);
    }
    for (; j<width ; ++j) {
        result.array[i][j] = array[i][j] * m.array[i][j];
    }
}
```

V mojom riešení ak je to možné načítavam 4x4 maticu a ju počítam so 4 SSE inštrukciami naraz. Ak je matica napr. 4x5, tak posledný stĺpec sa dokončí obyčajným násobením. Ak by matica bola napr. 5x5, v poslednom riadku prvé 4 hodnoty sa vypočítajú naraz, posledná je dokončená obyčajným násobením.

4 Záver

Kvôli môjmu nedostatku času som sa nemohol viac venovať tomuto projektu a tým pádom som celkom sklamaný zo svojho výsledku. Pri testovaní s 30 iteráciami som dosiahol zrýchlenie len o 1 sekundu, kvôli čomu som sem nevložil sekciu o testovaní. Určite by bolo možné korektne optimalizovať / spraviť nepotrebnou funkciu *transpose()* (o čo som sa aj pokúšal, ale nestihol som spojazdniť správne indexovanie a vkladanie do novej matice), pri ktorej sa kopírujú všetky dáta do novej matice. Ďalej šlo optimalizovať fukncie *sigmoid()*, na čo mi však bohužiaľ nezostal čas.

5 Referencie a zdroje

[1] https://www.researchgate.net/figure/Scalar-vs-SIMD-operation-for-multiple-additions_fig4_330140206