Assignment 7: SIMD Intrinsics

Speaker: Christoph Neuhauser

Technische Universität München (TUM)

Sequential Algorithm: Matrix-matrix Multiplication

```
for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
        for (int k = 0; k < n; k++){
            c[i * n + j] +=
                  a[i * n  + k] * b[j * n  + k];
        }
    }
}
```

Row-wise Row-wise

Algorithm & Pseudo-code

$$C = AB^T, c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{jk}$$

Use AVX2 to compute $\frac{256bit}{32bit} = 8$ multiplications of the dot product at once!

for all k in {0, 8, 16, …}:

sum += $\begin{pmatrix} a_{ik} \\ a_{i(k+1)} \\ \dots \\ a_{i(k+7)} \end{pmatrix} \circ \begin{pmatrix} b_{jk} \\ b_{j(k+1)} \\ \dots \\ b_{j(k+7)} \end{pmatrix}$

c_{ij} = sum

AVX2 Instructions

What functions we need:

`_mm256_loadu_ps:` Load 8 floating-point values to AVX2 vector register.

`_mm256_mul_ps:` Multiply two vectors element-wise.

`_mm256_add_ps:` Sum of two vectors element-wise.

What we unfortunately can‘t use:

`_mm256_dp_ps:` ”[…] performs a SIMD multiplication of the **lower four** packed single-precision floating-point elements […]”

-> Why does this even exist Intel?!

Implementation using AVX2

```
__m256 a_row, b_row, product, sum, hi;
assert(n % 8 == 0);

for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
        // “sum” will contain the cross-product
        // of a_row and b_row
        sum = _mm256_setzero_ps();
        for (int k = 0; k < n; k += 8){
            […]
        }
        […]
        // Access result
        c[i * n + j] = ((float*)&sum)[0];
    }
```

Implementation using AVX2

```
__m256 a_row, b_row, product, sum, hi;
assert(n % 8 == 0);

for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
        // "sum" will contain the cross-product
        // of a_row and b_row
        sum = _mm256_setzero_ps();
        for (int k = 0; k < n; k += 8){
            […]
        }
        […]
        // Access result
        c[i * n + j] = ((float*)&sum)[0];
    }
```

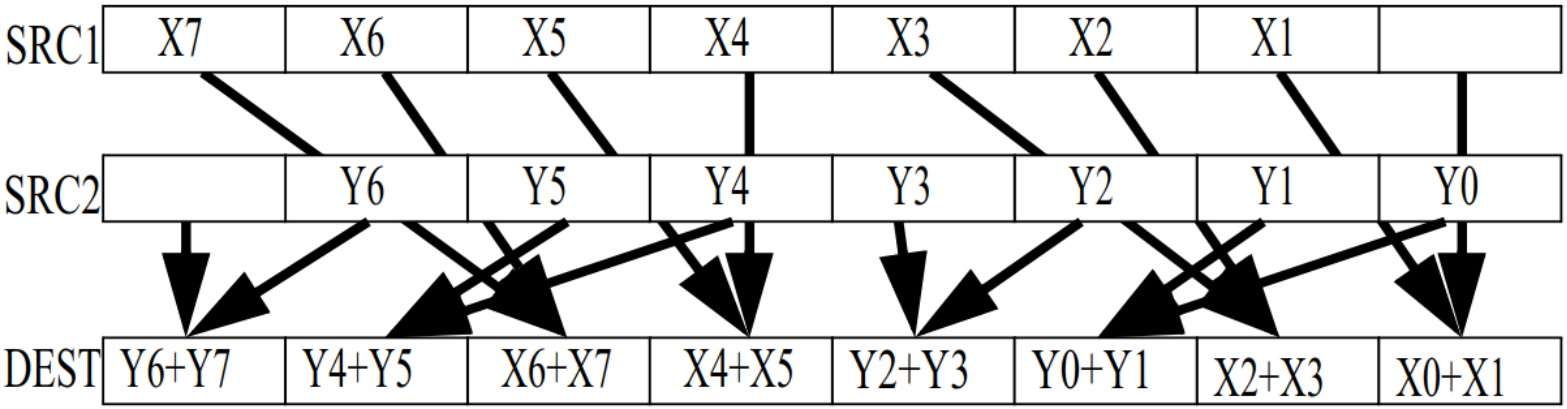
Implementation using AVX2

```
// Sum will contain the cross-product
// of a_row and b_row
sum = _mm256_setzero_ps();
for (int k = 0; k < n; k += 8){
    a_row = _mm256_loadu_ps(&a[i * n  + k]);
    b_row = _mm256_loadu_ps(&b[j * n  + k]);

    // Multiply and add. Unfortunately,_mm256_dp_ps
    // only supports four instead of eight floats :(
    product = _mm256_mul_ps(a_row, b_row);
    sum = _mm256_add_ps(sum, product);
}
```

Implementation using AVX2

```
// Now we need to horizontally sum the elements in sum
hi  = _mm256_permute2f128_ps(sum, sum, 1);
sum = _mm256_add_ps(sum, hi);
sum = _mm256_hadd_ps(sum, sum);
sum = _mm256_hadd_ps(sum, sum);
```




```
// Access result
c[i * n + j] = ((float*)&sum)[0];
```

Sequential vs. Optimized Algorithm

`Speedup: x4.2`

Unfortunately, coming close to x8.0 impossible without hardware support for 256-bit dot-product.

Sources

- Image on slide 8: https://www.felixcloutier.com/x86/HADDPS.html
- _mm256_dp_ps: https://software.intel.com/en-us/node/524056

Any questions?