MZT

Laboratorium 1

Przemysław Kleszcz

Nr albumu: 124624

Platforma testowa

Procesor	Intel Core i5-7440HQ, 2.8GHz, 6MB Cache
RAM	DDR4 - 16 GB
System operacyjny	Microsoft Windows 10 Pro
Środowisko programistyczne	Visual Studio Professional 2017 v15.5.7
Środowisko uruchomieniowe	.NET Framework 4.6.01055

Zadanie 1

```
void matvec_opt_1(double* a, double* x, double* y, int n)
/*=========================================================================
Usuniecie skokow w danych
===========================================================================*/
{
       int i, j = 0, k = 0;
       memset(y, 0, n * sizeof(double));
       double register r;
       for (j = 0; j<n; j++) {
              r = x[j];
              for (i = 0; i<n; i++) {
                     y[i] += a[k++] * r;
              }
       }

}
```

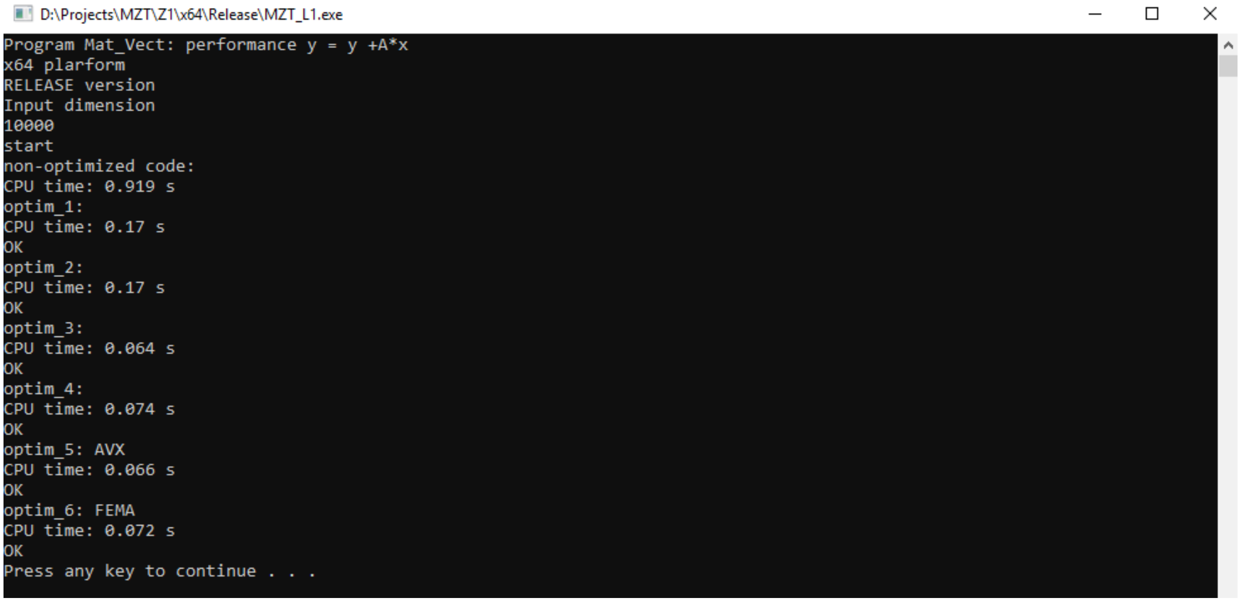
```
void matvec_opt_2(double* a, double* x, double* y, int n)
/*============================================================================
Rozwijanie petli
=============================================================================*/
{
        int rest = n % 8;
        int i = 0, j = 0, ij = 0;
        double register r = 0;

        memset(y, 0, n * sizeof(double));

        for (j = 0; j < n; ++j) {
                r = x[j];
                for (i = 0; i < n - rest; i += 8) {
                        y[i] += a[ij] * r;
                        y[i + 1] += a[ij + 1] * r;
                        y[i + 2] += a[ij + 2] * r;
                        y[i + 3] += a[ij + 3] * r;
                        y[i + 4] += a[ij + 4] * r;
                        y[i + 5] += a[ij + 5] * r;
                        y[i + 6] += a[ij + 6] * r;
                        y[i + 7] += a[ij + 7] * r;
                        ij += 8;
                }

                for (; i < n; ++i) {
                        y[i] += a[ij++] * r;
                }
        }

}
```



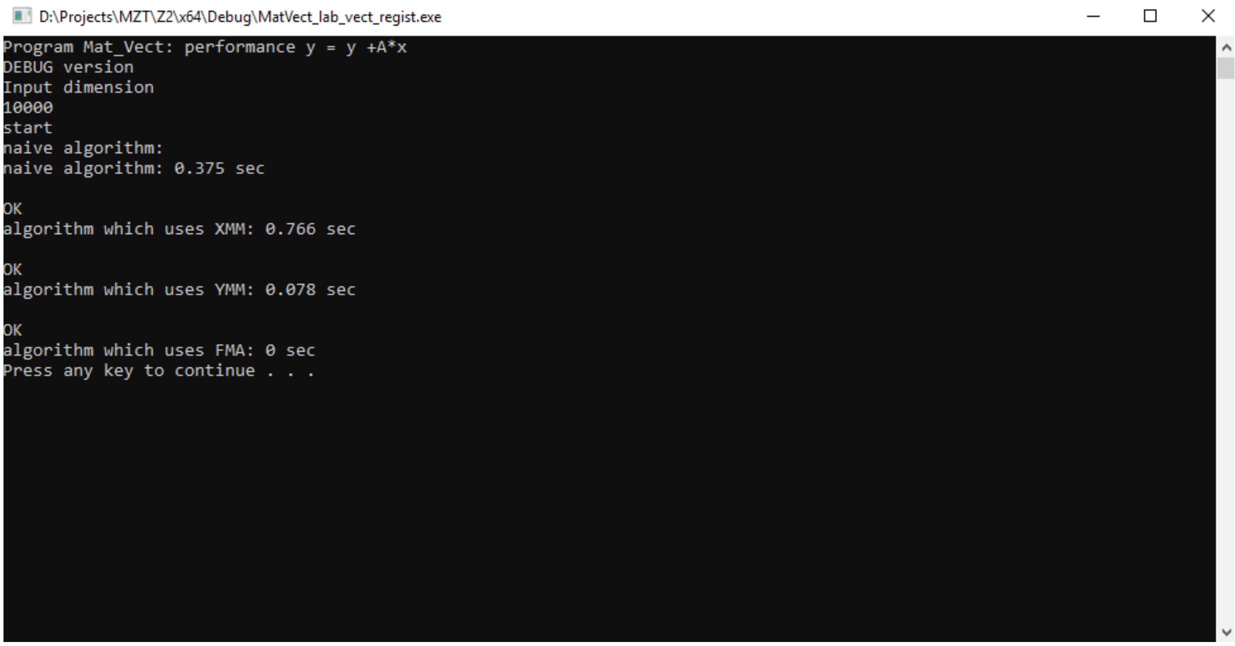
```
Program Mat_Vect: performance y = y +A*x
x64 plarform
RELEASE version
Input dimension
10000
start
non-optimized code:
CPU time: 0.919 s
optim_1:
CPU time: 0.17 s
OK
optim_2:
CPU time: 0.17 s
OK
optim_3:
CPU time: 0.064 s
OK
optim_4:
CPU time: 0.074 s
OK
optim_5: AVX
CPU time: 0.066 s
OK
optim_6: FEMA
CPU time: 0.072 s
OK
Press any key to continue . . .
```

Zadanie 2

```
void matvec_XMM(double* a, double* x, double* y, int n, int lb)
{
        int i, j;
        __m128d rx0, ra0, ry0;
        double *ptr_x, *ptr_a;
        __declspec(align(16)) double tmp[2];
        memset((void *)y, 0, n * sizeof(double));
        ptr_a = a;
        for (i = 0; i < n; i++)
        {
                ry0 = _mm_setzero_pd();
                ptr_x = x;
                for (j = 0; j < n; j += 2, ptr_a += 2, ptr_x += 2)
                {
                        rx0 = _mm_load_pd(ptr_x);
                        ra0 = _mm_load_pd(ptr_a);
                        ra0 = _mm_mul_pd(ra0, rx0);
                        ry0 = _mm_add_pd(ry0, ra0);
                }
                _mm_store_pd(tmp, ry0);
                y[i] = tmp[0] + tmp[1];
        }
}

void matvec_YMM(double* a, double* x, double* y, int n, int lb)
{
        int i, j;
        __m256d rx0, ra0, ra1, ra2, ra3, ry0, ry1, ry2, ry3;
        double *ptr_x, *ptr_a;
        __declspec(align(16)) double tmp0[4], tmp1[4], tmp2[4], tmp3[4];
        memset((void *)y, 0, n * sizeof(double));
        ptr_a = a;
        for (i = 0; i < n; i += 4)
        {
                ry0 = ry1 = ry2 = ry3 = _mm256_setzero_pd();
                ptr_x = x;
                for (j = 0; j < n; j += 16)
                {
                        _mm_prefetch((const char *)(ptr_x + 16), _MM_HINT_T0);
                        _mm_prefetch((const char *)(ptr_x + 24), _MM_HINT_T0);
                        _mm_prefetch((const char *)(ptr_a + 16), _MM_HINT_NTA);
                        _mm_prefetch((const char *)(ptr_a + 24), _MM_HINT_NTA);
                        //poprawic
                        //--------------------------0
                        rx0 = _mm256_load_pd(ptr_x);
                        ra0 = _mm256_load_pd(ptr_a);
                        ra1 = _mm256_load_pd(ptr_a + n);
                        ra2 = _mm256_load_pd(ptr_a + 2 * n);
                        ra3 = _mm256_load_pd(ptr_a + 3 * n);
                        ra0 = _mm256_mul_pd(ra0, rx0);
                        ra1 = _mm256_mul_pd(ra1, rx0);
                        ra2 = _mm256_mul_pd(ra2, rx0);
                        ra3 = _mm256_mul_pd(ra3, rx0);
                        ry0 = _mm256_add_pd(ry0, ra0);
                        ry1 = _mm256_add_pd(ry1, ra1);
```

```
            ry2 = _mm256_add_pd(ry2, ra2);
            ry3 = _mm256_add_pd(ry3, ra3);
            //-------256----------------1
            rx0 = _mm256_load_pd(ptr_x + 4);
            ra0 = _mm256_load_pd(ptr_a + 4);
            ra1 = _mm256_load_pd(ptr_a + n + 4);
            ra2 = _mm256_load_pd(ptr_a + 2 * n + 4);
            ra3 = _mm256_load_pd(ptr_a + 3 * n + 4);
            ra0 = _mm256_mul_pd(ra0, rx0);
            ra1 = _mm256_mul_pd(ra1, rx0);
            ra2 = _mm256_mul_pd(ra2, rx0);
            ra3 = _mm256_mul_pd(ra3, rx0);
            ry0 = _mm256_add_pd(ry0, ra0);
            ry1 = _mm256_add_pd(ry1, ra1);
            ry2 = _mm256_add_pd(ry2, ra2);
            ry3 = _mm256_add_pd(ry3, ra3);
            //-------256----------------2
            rx0 = _mm256_load_pd(ptr_x + 8);
            ra0 = _mm256_load_pd(ptr_a + 8);
            ra1 = _mm256_load_pd(ptr_a + n + 8);
            ra2 = _mm256_load_pd(ptr_a + 2 * n + 8);
            ra3 = _mm256_load_pd(ptr_a + 3 * n + 8);
            ra0 = _mm256_mul_pd(ra0, rx0);
            ra1 = _mm256_mul_pd(ra1, rx0);
            ra2 = _mm256_mul_pd(ra2, rx0);
            ra3 = _mm256_mul_pd(ra3, rx0);
            ry0 = _mm256_add_pd(ry0, ra0);
            ry1 = _mm256_add_pd(ry1, ra1);
            ry2 = _mm256_add_pd(ry2, ra2);
            ry3 = _mm256_add_pd(ry3, ra3);
            //-------256----------------3
            rx0 = _mm256_load_pd(ptr_x + 12);
            ra0 = _mm256_load_pd(ptr_a + 12);
            ra1 = _mm256_load_pd(ptr_a + n + 12);
            ra2 = _mm256_load_pd(ptr_a + 2 * n + 12);
            ra3 = _mm256_load_pd(ptr_a + 3 * n + 12);
            ra0 = _mm256_mul_pd(ra0, rx0);
            ra1 = _mm256_mul_pd(ra1, rx0);
            ra2 = _mm256_mul_pd(ra2, rx0);
            ra3 = _mm256_mul_pd(ra3, rx0);
            ry0 = _mm256_add_pd(ry0, ra0);
            ry1 = _mm256_add_pd(ry1, ra1);
            ry2 = _mm256_add_pd(ry2, ra2);
            ry3 = _mm256_add_pd(ry3, ra3);
            ptr_a += 16;
            ptr_x += 16;
        }
        ptr_a += 3 * n;
        _mm256_store_pd(tmp0, ry0);
        _mm256_store_pd(tmp1, ry1);
        _mm256_store_pd(tmp2, ry2);
        _mm256_store_pd(tmp3, ry3);
        y[i] = tmp0[0] + tmp0[1] + tmp0[2] + tmp0[3];
        y[i + 1] = tmp1[0] + tmp1[1] + tmp1[2] + tmp1[3];
        y[i + 2] = tmp2[0] + tmp2[1] + tmp2[2] + tmp2[3];
        y[i + 3] = tmp3[0] + tmp3[1] + tmp3[2] + tmp3[3];
    }
}
```



```
Program Mat_Vect: performance y = y +A*x
DEBUG version
Input dimension
10000
start
naive algorithm:
naive algorithm: 0.375 sec

OK
algorithm which uses XMM: 0.766 sec

OK
algorithm which uses YMM: 0.078 sec

OK
algorithm which uses FMA: 0 sec
Press any key to continue . . .
```