Optymalizacja Kodu Na Różne Architektury
Ćwiczenie 2

1 Wprowadzenie

Rozważmy następującą procedurę uruchamiającą faktoryzację Choleskiego dla macierzy gęstych

Listing 1: Bazowa wersja kodu

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>

#define IDX(i, j, n) (((j)+ (i)*(n)))



static double gtod_ref_time_sec = 0.0;

/* Adapted from the bl2_clock() routine in the BLIS library */

double dclock(){
    double the_time, norm_sec;
    struct timeval tv;
    gettimeofday( &tv, NULL );
    if ( gtod_ref_time_sec == 0.0 )
        gtod_ref_time_sec = ( double ) tv.tv_sec;
    norm_sec = ( double ) tv.tv_sec - gtod_ref_time_sec;
    the_time = norm_sec + tv.tv_usec * 1.0e-6;
    return the_time;
```

```
}


int chol(double * A, unsigned int n){
    int i, j, k;

    for (j = 0; j < n; j++) {
        for (i = j; i < n; i++) {
            for (k = 0; k < j; k++) {
                A[IDX(i, j, n)] -= A[IDX(i, k, n)] * A[IDX(j, k, n)];
            }
        }

        if (A[IDX(j, j, n)] < 0.0) {
            return (1);
        }

        A[IDX(j, j, n)] = sqrt(A[IDX(j, j, n)]);
        for (i = j + 1; i < n; i++){
            A[IDX(i, j, n)] /= A[IDX(j, j, n)];
        }
    }

    return (0);
}

int main(int argc, char ** argv){
    double* A;
    double t1, t2;
    int i, j, n, ret;
    double dtime;

    n = atoi(argv[1]);
    A = malloc(n * n * sizeof(double));
    assert(A != NULL);

    for (i = 0; i < n; i++) {
        A[IDX(i, i, n)] = 1.0;
    }

    dtime = dclock();
    if (chol(A, n)) {
```

```
        fprintf(stderr,"Matrix is not symmetric or not positive definite\n");
    }
    dtime = dclock()-dtime;
    printf( "Time: %le \n", dtime);

    fflush( stdout );

    free(A);
    return 0;
}
```

2 Ćwiczenia

Przy kolejnych krokach proszę wykonywać kopie aktualnych wersji stanu kodu. Będą potrzebne przy ostatnim punkcie oraz na kolejnych zajęciach. Proszę na bieżąco notować czasy wykonywania się obliczeń.

2.1

Proszę wprowadzić zaznaczone w kodzie modyfikacje a następnie skompilować procedurę bez optymalizacji. Parametr uruchomieniowy definiuje rozmiar macierzy.

```
maciekw@Banach:~/studenci/OORA/skrypty/lab2> gcc -lm chol1.c
maciekw@Banach:~/studenci/OORA/skrypty/lab2> ./a.out 4000
Time: 3.734360e+01
maciekw@Banach:~/studenci/OORA/skrypty/lab2>
```

2.2

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab2> cp chol1.c chol2.c
```

Proszę umieścić iteratory pętli w rejestrach oraz utworzyć w rejestrze zmienną z wartością często używaną w pętli.

```
register unsigned int i,j,k;
register double tmp;
...
tmp = A[IDX(i, j, n)];
for (k = 0; k < j; k++)
    tmp -= A[IDX(i, k, n)] * A[IDX(j, k, n)];
```

```
...
tmp = sqrt(A[IDX(j, j, n)]);
for (i = j + 1; i < n; i++)
    A[IDX(i, j, n)] /= tmp;
```

Proszę skompilować procedurę z optymalizacjami.
Jaki jest czas działania?

2.3

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab2> cp chol2.c chol3.c
```

Proszę wykonać ręczne rozwinięcie pętli o 8 iteracji, oraz procedurę **max(a,b)**.

```
for (k = 0; k < j; ) {
    if (k < max(j - BLKSIZE, 0)) {
        tmp -= A[IDX(i, k, n)] * A[IDX(j, k, n)] +
        ...; // PROSZE UZUPELNIC
        k += BLKSIZE;
    } else {
        tmp -= A[IDX(i, k, n)] * A[IDX(j, k, n)];
        k++;
    }
}
```

Proszę skompilować procedurę z optymalizacją.
Jaki jest czas działania?

2.4

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab2> cp chol3.c chol4.c
```

Proszę użyć operacji wektorowych.

```
#include <x86intrin.h>
...
register __m128d tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
...
tmp0 = _mm_loadu_pd(A+IDX(i, k, n));
/* tmp0 = (A[IDX(i,k,n)],A[IDX(i,k+1,n)]) */
tmp1 = _mm_loadu_pd(A+IDX(j, k, n));
/* tmp1 = (A[IDX(j,k,n)],A[IDX(j,k+1,n)]) */
tmp2 = _mm_loadu_pd(A+IDX(i, k+2, n));
```

```
/* tmp2 = (A[IDX(i,k+2,n)],A[IDX(i,k+3,n)]) */
tmp3 = _mm_loadu_pd(A+IDX(j, k+2, n));
/* tmp3 = (A[IDX(j,k+2,n)],A[IDX(j,k+3,n)]) */
//PROSZE UZUPELNIC
...
tmp0 = _mm_mul_pd(tmp0, tmp1); // multiply
tmp2 = _mm_mul_pd(tmp2, tmp3);
//PROSZE UZUPELNIC
...
tmp0 = _mm_add_pd(tmp0, tmp2); // add 2 vectors
...
tmp0 = _mm_add_pd(tmp0, tmp4);
tmp -= tmp0[0] + tmp0[1];
```

Proszę skompilować procedurę z optymalizacją.
Jaki jest czas działania?

2.5

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab2> cp chol4.c chol5.c
```

Proszę zwiększyć ręczne rozwinięcie pętli do 16 iteracji z wykorzystaniem operacji wektorowych. Proszę skompilować procedurę z optymalizacją.
Jaki jest czas działania?

2.6

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab2> cp chol5.c chol6.c
```

Proszę wymienić operacje wektorowe 128 bitowe na 256 bitowe

```
#include <immintrin.h>
...
register __m256d tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
...
tmp0 = _mm256_loadu_pd(A+IDX(i, k, n));
/*tmp0 = (A[IDX(i,k,n)],A[IDX(i,k+1,n)],A[IDX(i,k+2,n)],A[IDX(i,k+3,n)])*/
//PROSZE UZUPELNIC
...
tmp0 = _mm256_mul_pd(tmp0, tmp1); // multiply
//PROSZE UZUPELNIC
...
```

```
tmp0 = _mm256_add_pd(tmp0, tmp2); // add 2 vectors
//PROSZE UZUPELNIC
...
tmp -= tmp0[0] + tmp0[1] + tmp0[2] + tmp0[3];
```

Proszę skompilować procedurę z optymalizacją.

```
maciekw@Banach:~/studenci/OORA/skrypty/lab2> gcc -lm -mavx chol6.c
```

Jaki jest czas działania?

2.7

Proszę wszystkie uzyskane na Laboratoriach kody skompilować z opcją optymalizacji **-O2** oraz przetestować opcje **-march=native** oraz **-mfma**.

```
maciekw@Banach:~/lab2> gcc -lm -mavx -O2 chol6.c
maciekw@Banach:~/lab2> ./a.out 4000
Time: 4.874826e+00
maciekw@Banach:~/lab2> gcc -lm -mavx -O2 -march=native chol6.c
maciekw@Banach:~/lab2> ./a.out 4000
maciekw@Banach:~/lab2> gcc -lm -mavx -O2  -mfma chol6.c
maciekw@Banach:~/lab2> ./a.out 4000
maciekw@Banach:~/lab2> gcc -lm -mavx -O2 -march=native -mfma chol6.c
maciekw@Banach:~/lab2> ./a.out 4000
```

Jaki jest ich czas działania?

3 Podsumowanie

Które metody optymalizacji dały najlepsze rezultaty?
Dlaczego? Jakie mechanizmy za nimi stały?