Załącznik nr 1 – wyniki pomiaru czasu

Liczba wątkow	sekwencyjnie	blokowo	cyklicznie
1	0.147607	0.143328	0.138835
2	nd	0.069371	0.072790
3	nd	0.054495	0.047221
4	nd	0.036901	0.041422
5	nd	0.029209	0.037911
6	nd	0.026756	0.040720
7	nd	0.023075	0.036225
8	nd	0.022081	0.038304

Załącznik nr 2 – kod programu

```
#include<stdlib.h>
#include<stdio.h>
#include<pthread.h>
#include<math.h>
#include"pomiar_czasu.h"
#define ROZMIAR 1000000 // dokładność
#define LICZBA_W 1
#define PRZEDZIAL_A 0
#define PRZEDZIAL_B M_PI
typedef struct {
   int a;
   int b;
   int id;
} Przedzial;
pthread_mutex_t muteks;
pthread_t watki[LICZBA_W];
double local_suma[LICZBA_W]; // muszę zainicjalizować o wartosci 0
Przedzial p[LICZBA_W];
int indeksy[LICZBA_W];
double f(double x) {
   return sin(x);
}
double suma_trapez(double x1, double x2) {
   return (f(x1)+f(x2))*(x2-x1)/2;
}
double licz_calke_sekwencyjnie() {
   double dx = ((double)PRZEDZIAL_B - (double)PRZEDZIAL_A) / (double)ROZMIAR;
   double x1;
   int i = 0;
   double suma = 0;
   while(i < ROZMIAR) {
      x1 = (double)PRZEDZIAL_A + (double)i * dx;
      suma += suma_trapez(x1,x1 + dx);
      i++;
      if(i >= ROZMIAR) {
         break;
      }
   }
   return suma;
}
// Dokładność się poprawi jeśli będe miał 100% pewności że jestem na n-1 kroku iteracji i końcowa wartość to PRZEDZIAL_B
void * licz_calke_cyklicznie(int id) {
   double dx = ((double)PRZEDZIAL_B - (double)PRZEDZIAL_A) / (double)ROZMIAR;
   double x1, x2;
   int i = id;
   while(i < ROZMIAR) {
      x1 = (double)PRZEDZIAL_A + (double)i * dx;
```

```
        x2 = x1 + dx;
        local_suma[id] += suma_trapez(x1,x2);
        i += LICZBA_W;
        if(i >= ROZMIAR) {
            break;
        }
    }
    pthread_exit( (void *)0);
}
void * licz_calke_blokowo(void *arg_wsk) {
    Przedzial p = *((Przedzial*) arg_wsk);
    double dx = ((double)PRZEDZIAL_B - (double)PRZEDZIAL_A) / (double)ROZMIAR;
    double x1, x2;
    int i = p.a;
    while(i < p.b) {
        x1 = (double)PRZEDZIAL_A + (double)i * dx;
        x2 = x1 + dx;
        local_suma[p.id] += suma_trapez(x1,x2);
        i++;
        if(i >= ROZMIAR) {
            break;
        }
    }
}
int main() {
    //Inicjalizacja obliczeń sekwencyjnych
    int it;
    printf("Rozpoczecie obliczen dla %d watkow i dokladnosci %d\n", LICZBA_W, ROZMIAR);
    printf("Obliczenia cyklicznie\n"); // Dekompozycja sterowania
    double suma = 0;
    inicjuj_czas();
    for(it = 0; it < LICZBA_W; it++) {
        indeksy[it] = it;
        local_suma[it] = 0;
        pthread_create( &watki[it], NULL, licz_calke_cyklicznie, (void *) indeksy[it] );
    }
    for(it = 0; it < LICZBA_W; it++) {
        pthread_join( watki[it], NULL ); //Kappa
        suma += local_suma[it];
    }
    drukuj_czas();
    printf("Koniec obliczen, suma wynosi %lf\n\n", suma);

    printf("Obliczenia blokowo\n"); // Dekompozycja blokowa
    suma = 0;
    inicjuj_czas();
    for(it = 0; it < LICZBA_W; it++) {
        local_suma[it] = 0;
        p[it].id = it;
        p[it].a = it * ROZMIAR / LICZBA_W;
        p[it].b = (it+1) * ROZMIAR / LICZBA_W;
        if(p[it].b > ROZMIAR) {
            p[it].b = ROZMIAR;
        }
        pthread_create( &watki[it], NULL, licz_calke_blokowo, (void *) &p[it]);
    }
    for(it = 0; it < LICZBA_W; it++) {
        pthread_join( watki[it], NULL );
        suma += local_suma[it];
    }
    drukuj_czas();
    printf("Koniec obliczen, suma wynosi %lf\n\n", suma);
    printf("Obliczenia sekwencyjne\n");
    inicjuj_czas();
    suma = licz_calke_sekwencyjnie();
    drukuj_czas();
    printf("Koniec obliczen, suma wynosi %lf\n\n", suma);
}
```