Laboratorium 4

Architektura Systemów Komputerowych
Praca potokowa procesorów

Uwaga! W ramach poniższych zadań należy przygotować sprawozdanie w formie pliku pdf wysłanego przez platformę e-learning. Sprawozdanie będzie stanowiło główną podstawę do oceny z laboratorium.

Zadanie 1 – projekt do samodzielnej realizacji

Przeanalizuj poniższy kod pod kątem wykorzystania potoków. Zaproponuj modyfikacje kodu w celu lepszego wykorzystania potoków poprzez rozpisanie najbardziej wewnętrznej części pętli for:

- z użyciem 2 buforów dla tymczasowych wyników pośrednich **s1 i s2**
- z użyciem 4 buforów dla tymczasowych wyników pośrednich **s1, s2, s3, i s4**
- z użyciem 8 buforów dla tymczasowych wyników pośrednich **s1, s2, s3, …, i s8**
- z użyciem 16 buforów dla tymczasowych wyników pośrednich **s1, s2, s3, …, i s16**

Pomoc do zadania 1 (link do nagrania)

Wersja v1 programu z jednym buforem „s1”:

```
#include <iostream>
#include <iomanip>
#include <omp.h>

int main() {

	const int n =1024; // rozmiar tablicy
	double * tab = new double[n]; // tworzenie obietu w pamięci dla tablicy dynamicznej

	for(int i=0; i<n; ++i) tab[i] = 0.0625; //wypełnianie tablicy


	double time; // zmienne dla pomiaru czasu

	double s1 = 0.0; //bufor dla sumy wszystkich elementów tablicy

	time = omp_get_wtime(); //pomiar czasu - start

	//poniższy fragment należy zmienić zgodnie z koncepcją z nagrania
	for(int t=0; t<n*n; ++t) {
		for(int i=0; i<n; ++i) {
			s1+=tab[i];
		}
	}
	time = omp_get_wtime() - time; //pomiar czasu - stop

	std::cout<<std::fixed;
	std::cout<<"1. suma = "<<std::setprecision(30)<<s1<<" ";
	std::cout<<"czas wykonania = "<<std::setprecision(3)<<time<<std::endl;

	delete [] tab; //kasowanie tablicy
	return 0;
}
```

Aby rozwiązać zadanie 1 należy:

- przygotować pięć wersji dla powyższego programu z użyciem różnej liczby buforów
- wszystkie wersje kodu należy skompilować i uruchomić, przykładowo:
  kompilacja: *g++ nazwa-v1.cpp -o nazwa-v1 -fopenmp -O1*
  uruchomienie: *./nazwa-v1*
- upewnić się, że wynik operacji sumowania wszystkich elementów tablicy jest poprawny i wynosi: *67108864.000000000000000000000000000000*
- zmierzyć czas wykonania zadania:

Wersja kodu	Liczba buforów s1, s2, …	Czas programu [s]
v1 (kod powyżej)	1	
v2	2	
v3	4	
v4	8	
v5	16	

W sprawozdaniu należy opisać:

- która wersja programu wykazała się najlepszym czasem (wypełnić powyższą tabelkę, a dla każdej wersji programu przeprowadzić kilka testów i wybrać najlepszy czas)
- wyjaśnić skąd wynika różnica
- zilustrować przebieg wybranej wersji programu realizację niewielkiej liczby iteracji na potkach procesora przy założeniu, że operacja dodawania zajmuje 4 cykle procesora

Zadanie 2 – projekt do samodzielnej realizacji

Dla każdej wersji powyższego programu zmień sposób inicjalizacji danych, a następnie wypełnij poniższą tabelę:

```
~~for(int i=0; i<n; ++i) tab[i] = 0.0625; //wypełnianie tablicy~~
 for(int i=0; i<n; ++i) tab[i] = 0.1; //wypełnianie tablicy
```

Wersja kodu	Liczba buforów s1, s2, …	Suma wszystkich elementów:
v1	1	107374180.705882295966148376464843750000
v2	2	
v3	4	
v4	8	
v5	16	

W sprawozdaniu należy opisać, dla której wersji programu wynik jest „bardziej” dokładny oraz spróbować wyjaśnić z czego wynika różnica.

Zadanie 3 – projekt do samodzielnej realizacji - wymagane na ocenę bdb

Przeanalizuj poniższy kod pod kątem wykorzystania potoków. Zaproponuj modyfikacje kodu (**niebieski fragment kodu**) w celu lepszego wykorzystania potoków.

Wersja v1 programu:

```
#include <iostream>
#include <stdlib.h>
#include <omp.h>

int main() {

  // Utworzenie tablic
  const int n =1024*1024*128;
  double * A = new double[n];
  double * B = new double[n];
  double * C = new double[n];
  double * D = new double[n];

  double time;

  // Inicjalizacja danych
  for(int i=0; i<n; ++i) {
    A[i]=1.0*i;
    B[i]=2.0*i;
    C[i]=3.0*i;
    D[i]=4.0*i;
  }

  time = omp_get_wtime(); // pomiar czasu - start

  // Fragment do modyfikacji
  for(int i=0; i<n; ++i) {
    A[i] = A[i]+B[i];
    C[i] = A[i]+1.0;
    A[i] = D[i]+1.0;
  }

  time = omp_get_wtime() - time; // pomiar czasu - stop


  std::cout<<" czas wykonania = "<<time;
  std::cout<<"    przykładowa wartość "<<A[3]<<std::endl;

  // Kasowanie danych
  delete [] A;
  delete [] B;
  delete [] C;
  delete [] D;

  return 0;
```

Aby rozwiązać zadanie 3 należy:

- zastanowić się w jaki sposób realizowane jest najbardziej zagnieżdżona część pętli zaznaczona w kodzie na niebiesko:
    - rozpisać zależności między instrukcjami w formie drzewa tak jak na nagraniu (tylko fragment kodu zaznaczony na niebiesko)
    - zilustrować ich wykonanie na potku zakładając, że operacja dodawania zajmuje np. 4 cykle procesora dla kilku iteracji pętli
- spróbować zmodyfikować niebieski fragment kodu poprzez wyeliminowanie zależności między instrukcjami (można wprowadzić np. zmienną pomocniczą do przechowywanie pośrednik wyników:
    - rozpisać zależności między instrukcjami w formie drzewa tak jak na nagraniu (tylko zmodyfikowany fragment kodu zaznaczony na niebiesko)
    - zilustrować ich wykonanie na potku zakładając, że operacja dodawania zajmuje np. 4 cykle procesora
- skompilować i uruchomić obie wersje kodu, przykładowo:
  kompilacja: *g++ nazwa-v1.cpp -o nazwa-v1 -fopenmp -O1*
  uruchomienie: *./nazwa-v1*
- planowanym efektem modyfikacji powinno być nieznaczne skrócenie czasu obliczeń, z czasu ok 0.438 [s] dla 1 wersji do czasu ok 0.406 [s] dla drugiej wersji
    - każdą wersję uruchomić ok 3-5 razy i wybrać najlepszy czas do porównania

W sprawozdaniu należy uwzględnić powyższe wytyczne.