Laboratorium 5a

Architektura Systemów Komputerowych
Praca jednostek FPU/VPU

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.

Pomoc do instrukcji:
Instrukcja składa się z kilku zadań, których celem jest stworzenie jednego programu. Proszę wstępnie przeczytać całą instrukcję, a następnie realizować zadanie po zadaniu.

Zadanie 1

W pierwszym zadaniu należy stworzyć funkcję w języku c++ realizującą dodawanie skalarne dwóch liczb typu double. Aby zrealizować to zadanie należy utworzyć dwa pliki w swoim katalogu o nazwach **f1.h** oraz **f1.cpp**

W pliku o nazwie **f1.h** należy wpisać deklaracje funkcji:

```
void scalar_add(double &t0, double &t1, double &t2);
```

, natomiast w pliku **f1.cpp** należy dodać definicję funkcji w postaci:

```
#include "f1.h"
void scalar_add(double &t0, double &t1, double &t2) {
     t2 = t0 + t1;
}
```

W następnym kroku należy skompilować powyższy program w celu wygenerowania kodu w asemblerze powyższej funkcji korzystając z następującej komendy:

g++ -S f1.cpp -O3 -fno-tree-vectorize

Flaga kompilatora gcc ***-fno-tree-vectorize*** wstrzymuje proces automatycznej wektoryzacji obliczeń.

W rezultacie powyższej kompilacji zostanie wygenerowany plik o nazwie **f1.s**, sprawdź jego zawartość, następnie znajdź nazwy:

- instrukcji odpowiedzialnych za dodawanie skalarne (w nazwie instrukcji będzie „add”)
- instrukcji odpowiedzialnych za kopiowanie danych do rejestrów (w nazwie będzie „mov”
- rejestrów odpowiedzialnych za przechowywanie danych (nazwa rejestru zazwyczaj znajduje się na końcu linijki z instrukcją np. ***addsd(%rsi), %xmm0****, gdzie xmm0 to poszukiwana nazwa rejestru)*

W ostatnim kroku assembluj kod zdefiniowanej funkcji do kodu maszynowego platformy docelowej kompilując program w następujący sposób:

g++ -c f1.cpp -O3 -fno-tree-vectorize

Otrzymany plik o nazwie **f1.o** zostanie wykorzystany w ostatnim zadaniu tej instrukcji

Zadanie 2

W drugim zadaniu należy stworzyć kolejną funkcję w języku c++ realizującą dodawanie wektorwe liczb typu double z wykorzystaniem standardu SSE. Aby zrealizować to zadanie należy utworzyć dwa pliki w swoim katalogu o nazwach **f2.h** oraz **f2.cpp**

W pliku o nazwie **f2.h** należy wpisać deklaracje funkcji:

```
#include <emmintrin.h>
void sse_add(__m128d &t0, __m128d &t1, __m128d &t2);
```

, natomiast w pliku **f2.cpp** należy dodać definicję funkcji w postaci:

```
#include "f2.h"

void sse_add(__m128d &t0, __m128d &t1, __m128d &t2) {
      t2 = _mm_add_pd ( t0, t1);
}
```

W następnym kroku należy skompilować powyższy program w celu wygenerowania kodu w asemblerze powyższej funkcji korzystając z następującej komendy:

g++ -S f2.cpp -O3 -mavx

Flaga kompilatora gcc **-mavx** umożliwia proces wektoryzacji zgodnie z Intel SSE

W rezultacie powyższej kompilacji zostanie wygenerowany plik o nazwie **f2.s**, sprawdź jego zawartość, następnie znajdź nazwy:

- instrukcji odpowiedzialnych za dodawanie skalarne (w nazwie instrukcji będzie „add”)
- instrukcji odpowiedzialnych za kopiowanie danych do rejestrów (w nazwie będzie „mov”
- rejestrów odpowiedzialnych za przechowywanie danych (nazwa rejestru zazwyczaj znajduje się na końcu linijki z instrukcją np. ***vaddpd (%rdi), %xmm0, %xmm0****, gdzie xmm0 to poszukiwana nazwa rejestru)*

W ostatnim kroku assembluj kod zdefiniowanej funkcji do kodu maszynowego platformy docelowej kompilując program w następujący sposób:

g++ -c f2.cpp -O3 -mavx

Otrzymany plik o nazwie **f2.o** zostanie wykorzystany w ostatnim zadaniu tej instrukcji.

Zadanie 3

W trzecim zadaniu należy stworzyć kolejną funkcję w języku c++ realizującą dodawanie wektorwe liczb typu double z wykorzystaniem standardu AVX. Aby zrealizować to zadanie należy utworzyć dwa pliki w swoim katalogu o nazwach **f3.h** oraz **f3.cpp**

W pliku o nazwie **f3.h** należy wpisać deklaracje funkcji:

```
#include <immintrin.h>
void avx_add(__m256d &t0, __m256d &t1, __m256d &t2);
```

, natomiast w pliku **f2.cpp** należy dodać definicję funkcji w postaci:

```
#include "f3.h"
void avx_add(__m256d &t0, __m256d &t1, __m256d &t2) {
      t2 = _mm256_add_pd ( t0, t1);
}
```

W następnym kroku należy skompilować powyższy program w celu wygenerowania kodu w asemblerze powyższej funkcji korzystając z następującej komendy:

g++ -S f3.cpp -O3 -mavx

Flaga kompilatora gcc **-mavx** umożliwia proces wektoryzacji zgodnie z Intel AVX

W rezultacie powyższej kompilacji zostanie wygenerowany plik o nazwie **f3.s**, sprawdź jego zawartość, następnie znajdź nazwy:

- instrukcji odpowiedzialnych za dodawanie skalarne (w nazwie instrukcji będzie „add”)
- instrukcji odpowiedzialnych za kopiowanie danych do rejestrów (w nazwie będzie „mov”
- rejestrów odpowiedzialnych za pobranie danych (nazwa rejestru zazwyczaj znajduje się na końcu linijki z instrukcją np. ***vaddpd (%rdi), %ymm0, %ymm0****, gdzie ymm0 to poszukiwana nazwa rejestru)*

W ostatnim kroku assembluj kod zdefiniowanej funkcji do kodu maszynowego platformy docelowej kompilując program w następujący sposób:

g++ -c f3.cpp -O3 -mavx

Otrzymany plik o nazwie **f3.o** zostanie wykorzystany w ostatnim zadaniu tej instrukcji.

Zadanie 4

Porównaj i znajdź różnice w zawartość plików z kodem w asemblerze wszystkich trzech funkcji wygenerowanych w trakcie kompilacji:

g++ -S f1.cpp -O3 -fno-tree-vectorize

g++ -S f2.cpp -O3 -mavx

g++ -S f3.cpp -O3 -mavx

Efekty porównania zamieść w sprawozdaniu koncentrując się na:

- rodzaju instrukcji odpowiedzialnych za dodawanie skalarne
- rodzaju instrukcji odpowiedzialnych za kopiowanie danych do rejestrów
- rodzaju rejestrów odpowiedzialnych za przechowywanie danych

Zadanie 5

Z wykorzystaniem poniższego programu napisanego w pliku o nazwie **main.cpp** porównaj czas wykonania obliczeń z użyciem wcześniej zdefiniowanych funkcji. Wyniki i komentarz do otrzymanych wyników dla porównania różnych funkcji zamieść w sprawozdaniu.

Kod programu oraz sposób kompilacji i uruchomienia na następnej stronie.

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

#include "f1.h"
#include "f2.h"
#include "f3.h"

int main() {
	const int n =1024*1024*1024;

	double *t0 = (double*)aligned_alloc(64, n*sizeof(double));
	double *t1 = (double*)aligned_alloc(64, n*sizeof(double));
	double *t2 = (double*)aligned_alloc(64, n*sizeof(double));

	for(int i=0; i<n; ++i) {
		t0[i] = i * 1.0;
		t1[i] = i*3.0;
		t2[i] = 0.0;
	}

	double time;

	time = omp_get_wtime();

	for(int i=0; i<n; ++i)
		scalar_add(t0[i],t1[i],t2[i]);

	time = omp_get_wtime() - time;
	std::cout<<time<<" czas wykonania obl. skalarnych"<<std::endl;

	time = omp_get_wtime();

	for(int i=0; i<n; i+=2)
		sse_add(*(__m128d*)&t0[i],*(__m128d*)&t1[i],*(__m128d*)&t2[i]);

	time = omp_get_wtime() - time;
	std::cout<<time<<" czas wykonania obl. wektorowych see"<<std::endl;

	time = omp_get_wtime();

	for(int i=0; i<n; i+=4)
		avx_add(*(__m256d*)&t0[i],*(__m256d*)&t1[i],*(__m256d*)&t2[i]);

	time = omp_get_wtime() - time;
	std::cout<<time<<" czas wykonania obl. wektorowych avx"<<std::endl;

	free(t0);
	free(t1);
	free(t2);

	return 0;
}
```

Kompilacja:

g++ -c f1.cpp -O3 -fno-tree-vectorize

g++ -c f2.cpp -O3 -mavx

g++ -c f3.cpp -O3 -mavx

g++ f*.o main.cpp -o exe -O0 -fopenmp

Uruchomienie:

./exe