Politechnika Wrocławska

Wstęp do wysokowydajnych komputerów

Laboratorium nr. 5

Wykonawca:	
Imię i Nazwisko, nr indeksu, wydział	**Katarzyna Idasz** Wydział Informatyki i Telekomunikacji (W4N)
Termin zajęć: dzień tygodnia, godzina	Wtorek (11:15 – 13:00)
Numer grupy	1
Data wykonania ćwiczenia	28.05.2024 r.
Data oddania sprawozdania	11.06.2024 r.
Ocena końcowa	

Spis treści

1. Wstęp

1.1. Cel zadania

Należało zaimplementować całkę oznaczona (1) - taką samą jak na poprzednich zajęciach - z wykorzystaniem przetwarzania wektorowego (SSE/AVX - PACKED).

$$\int_a^b \frac{x^4 - x}{1 - 3x} dx \tag{1}$$

Granica pozostała niezmieniona względem poprzednich zajęć.
Następnie należało:

A. zmierzyć wydajność w GFLOPS oraz w IPC - *instructions per cycle*
B. wprowadzić optymalizację
C. porównać wyniki z kodem dla x87

2. Kody

2.1. Kod w C

Tak jak na poprzednich zajęciach, zaimplementowano metodę lewych prostokątów.
Długość przedziału ustawiono na dx = 0.0001.
Liczona jest całka $\int_1^5 f(x)\, dx$.
Aby wykorzystać przetwarzanie wektorowe użyto biblioteki *<immintrin.h>*.
W trakcie jednej iteracji pętli wykonywane są obliczenia dla czterech kolejnych wartościach x, w przeciwieństwie do implementacji całki w x87, gdzie obliczana była jedna wartość.
Kod został napisany po zajęciach.

```
#include <stdio.h>
#include <immintrin.h>

double dx = 0.0001;
double result = 0.0;

int main() {
	__m256d dx_vec = _mm256_set1_pd(dx);
	__m256d one_vec = _mm256_set1_pd(1.0);
	__m256d three_vec = _mm256_set1_pd(3.0);
	__m256d result_vec = _mm256_setzero_pd();

	for(double x = 1.0; x <= 5.0; x += 4*dx) {
		__m256d x_vec = _mm256_set_pd(x + 3*dx, x + 2*dx, x + dx, x);
```

```
        __m256d x2_vec = _mm256_mul_pd(x_vec, x_vec); // x*x
        __m256d x3_vec = _mm256_mul_pd(x2_vec, x_vec); // x*x*x
        __m256d x4_vec = _mm256_mul_pd(x3_vec, x_vec); // x*x*x*x

        // (x*x*x*x - x)
        __m256d first_vec = _mm256_sub_pd(x4_vec, x_vec);

        // (1.0 - 3.0*x)
        __m256d second_vec = _mm256_sub_pd(one_vec, _mm256_mul_pd(three_vec, x_vec));

        // (x*x*x*x - x)/(1.0 - 3.0*x) * dx
        __m256d result_i_vec = _mm256_mul_pd(_mm256_div_pd(first_vec, second_vec), dx_vec);

        // Dodanie do wyniku
        result_vec = _mm256_add_pd(result_vec, result_i_vec);
    }

    double result_array[4];
    _mm256_storeu_pd(result_array, result_vec);

    for (int i = 0; i < 4; i++) {
        result += result_array[i];
    }


    return 0;
}
```

Kod 1. nowacalka.c

2.2. Kod asemblerowy

Aby wygenerować kod użyto:

$ gcc -O2 -mavx2 -mfma -S nowacalka.c

```
    .file   "nowacalka.c"
    .text
    .section    .text.startup,"ax",@progbits
    .p2align 4
    .globl  main
    .type   main, @function
main:
.LFB5512:
    .cfi_startproc
    endbr64
    vmovsd  dx(%rip), %xmm4
```

```
    vmulsd    .LC1(%rip), %xmm4, %xmm9
    vxorpd    %xmm3, %xmm3, %xmm3
    vmulsd    .LC2(%rip), %xmm4, %xmm8
    vmovsd    .LC0(%rip), %xmm2
    vaddsd    %xmm4, %xmm4, %xmm10
    vmovapd    .LC3(%rip), %ymm7
    vmovapd    .LC4(%rip), %ymm6
    vbroadcastsd    %xmm4, %ymm11
    vmovsd    .LC5(%rip), %xmm5
    .p2align 4,,10
    .p2align 3
.L2:
    vaddsd    %xmm2, %xmm4, %xmm0
    vaddsd    %xmm2, %xmm10, %xmm1
    vaddsd    %xmm2, %xmm9, %xmm12
    vunpcklpd    %xmm0, %xmm2, %xmm0
    vaddsd    %xmm8, %xmm2, %xmm2
    vunpcklpd    %xmm12, %xmm1, %xmm1
    vinsertf128    $0x1, %xmm1, %ymm0, %ymm0
    vmulpd    %ymm0, %ymm0, %ymm1
    vcomisd    %xmm2, %xmm5
    vmulpd    %ymm0, %ymm1, %ymm1
    vfmsub132pd    %ymm0, %ymm0, %ymm1
    vfnmadd132pd    %ymm7, %ymm6, %ymm0
    vdivpd    %ymm0, %ymm1, %ymm0
    vfmadd231pd    %ymm11, %ymm0, %ymm3
    jnb    .L2
    vmovsd    result(%rip), %xmm0
    vunpckhpd    %xmm3, %xmm3, %xmm1
    xorl    %eax, %eax
    vaddsd    %xmm3, %xmm0, %xmm0
    vextractf128    $0x1, %ymm3, %xmm3
    vaddsd    %xmm1, %xmm0, %xmm0
    vaddsd    %xmm3, %xmm0, %xmm0
    vunpckhpd    %xmm3, %xmm3, %xmm3
    vaddsd    %xmm3, %xmm0, %xmm0
    vmovsd    %xmm0, result(%rip)
    vzeroupper
    ret
    .cfi_endproc
.LFE5512:
    .size    main, .-main
    .globl    result
    .bss
    .align 8
    .type    result, @object
```

```
    .size    result, 8
result:
    .zero    8
    .globl    dx
    .data
    .align 8
    .type    dx, @object
    .size    dx, 8
dx:
    .long    -350469331
    .long    1058682594
    .set    .LC0,.LC4
    .set    .LC1,.LC3
    .section    .rodata.cst8,"aM",@progbits,8
    .align 8
.LC2:
    .long    0
    .long    1074790400
    .section    .rodata.cst32,"aM",@progbits,32
    .align 32
.LC3:
    .long    0
    .long    1074266112
    .long    0
    .long    1074266112
    .long    0
    .long    1074266112
    .long    0
    .long    1074266112
    .align 32
.LC4:
    .long    0
    .long    1072693248
    .long    0
    .long    1072693248
    .long    0
    .long    1072693248
    .long    0
    .long    1072693248
    .section    .rodata.cst8
    .align 8
.LC5:
    .long    0
    .long    1075052544
    .ident    "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
    .section    .note.GNU-stack,"",@progbits
```

```
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string    "GNU"
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align 8
4:
```

Kod 2. nowacalka.s

3. Wydajność

3.1. GFLOPS

$$n = \frac{end-start}{precision} = \frac{5-1}{4 \cdot 0{,}0001} = 10\,000 \quad (2)$$

$$P = \frac{n \cdot N_A}{t} = \frac{10\,000 \cdot 9 \cdot 4}{0{,}001393120} \approx 258\,412\,771 \approx 0,258\,GFLOPS \quad (3)$$

$N_A -$ $liczba\ wykonanych\ operacji\ (arytmetycznych)$

$n -$ $ilość\ przejść\ pętli$

$t -$ $czas\ w\ [s]$

Do wzoru (2) podstawiono wartości, które zostały wykorzystane w kodzie (1). Do obliczenia czasu w równaniu (3) wykorzystano *perf stat*. Wykonano 20 pomiarów - tabela (1). Wyniki były zbliżone - brak odstających wartości - dlatego do wzoru podstawiono ich średnią.

Tabela 1. Czas w [s] wykoanania programu

Nr pomiaru	Czas [s]
1	0,001380732
2	0,001396740
3	0,001400132
4	0,001389537
5	0,001400480
6	0,001401149
7	0,001408425
8	0,001408707
9	0,001390348
10	0,001404097
11	0,001406530
12	0,001390877
13	0,001391600
14	0,001403575
15	0,001381770
16	0,001368485
17	0,001376113
18	0,001387099
19	0,001370706
20	0,001405295
Średnia	0,001393120

```
Performance counter stats for './program':

            0,48 msec task-clock                #    0,350 CPUs utilized
               0      context-switches          #    0,000 /sec
               0      cpu-migrations            #    0,000 /sec
              52      page-faults               #  107,587 K/sec
       1 880 338      cycles                    #    3,890 GHz
         167 171      stalled-cycles-frontend   #    8,89% frontend cycles idle
         330 200      stalled-cycles-backend    #   17,56% backend cycles idle
       1 222 648      instructions              #    0,65  insn per cycle
                                                #    0,27  stalled cycles per insn
         232 514      branches                  #  481,066 M/sec
   <not counted>      branch-misses                                                 (0,00%)

     0,001380732 seconds time elapsed

     0,001460000 seconds user
     0,000000000 seconds sys
```

Obraz 1. Przykład perf stat

3.2. IPC

$$P = \frac{n \cdot N_F}{C} = \frac{10\,000 \cdot 14 \cdot 4}{310657{,}5} \approx 1{,}8 \text{ IPC} \qquad (4)$$

$N_F -$ *liczba wykonanych operacji zmiennoprzecinkowych*
$n -$ *ilość przejść pętli*
$C -$ *ilość cykli*

Do obliczenia ilości cykli w równaniu (4) wykorzystano *rdtsc* - kod z poprzednich zajęć - time.s. Odczytano liczbę cykli przed wejściem do pętli oraz po wyjściu z niej, a następnie odjęto te wartości od siebie (analogicznie jak na poprzednich laboratoriach).
Wykonano 20 pomiarów - tabela (2). Wyniki były zbliżone - brak odstających wartości - dlatego do wzoru podstawiono ich średnią.

Tabela 2. Ilość cykli

Nr pomiaru	Ilość cykli
1	308430
2	309060
3	312210
4	308520
5	308700
6	312150
7	308970
8	312270
9	312270
10	312360
11	312000
12	308970
13	312120
14	308640
15	312180
16	312210
17	308490
18	312690
19	312150
20	308760
Średnia	310657,5

```
.global readTime
.type readTime, @function

readTime:
    push %ebx
    xor %eax, %eax
    cpuid
    rdtsc
    pop %ebx
    ret
```

Kod 3. time.s

4. Porównanie x87 i SSE/AVX

4.1. Średni czas [s]

Przetwarzanie wektorowe jest średnio

$$\frac{t_{x87}}{t_{SSE/AVX}} = \frac{0{,}002557632}{0{,}001393120} \approx 1,8$$ razy szybsze.

4.2. Średnia ilość cykli

Przetwarzanie wektorowe wymaga średnio

$$\frac{C_{x87}}{C_{SSE/AVX}} = \frac{612836}{310657{,}5} \approx 1,97$$ razy mniej cykli.

4.3. GFLOPS

W przypadku przetwarzania wektorowego wydajność w GFLOPS-ach jest

$$\frac{P_{SSE/AVX}}{P_{x87}} = \frac{0{,}258}{0{,}156} \approx 1,65$$ razy większa.

4.4. IPC

W przypadku przetwarzania wektorowego wydajność w IPC jest

$$\frac{P_{SSE/AVX}}{P_{x87}} = \frac{1{,}8}{1{,}57} \approx 1,15$$ razy większa.