SPRAWOZDANIE Z LABORATORIUM 4

Organizacja i architektura komputerów

mgr inż. Tomasz Serafin
Prowadzący laboratorium

Kamil Hulewicz
Numer albumu: 248889

Wstęp

W ramach zajęć laboratoryjnych należało zaimplementować proste działania arytmetyczne wykorzystując mechanizmy SISD (ang. Single Input Single Data) oraz SIMD (ang. Single Input Multiple Data). Następnie należało dokonać pomiarów czasu, jaki był wymagany do realizacji działań dla zestawów 2048, 4096 oraz 8192 liczb. Na podstawie otrzymanych wyników należało przeprowadzić analizę oraz zaprezentować zysk otrzymany w wyniku zastosowania mechanizmu SIMD.

Przebieg prac

Początkowo przystąpiono do implementacji 128 – bitowych struktur danych przechowujących liczby zmiennoprzecinkowe pojedynczej precyzji oraz wypełniania wspomnianych struktur danymi losowymi. W celu uniknięcia konieczności generowania wielu struktur o różnych długościach zdecydowano się na wygenerowanie pojedynczego zestawu struktur zawierających łącznie 8192 liczby, gdyż można na nich przeprowadzić wszystkie konieczne działania. Implementację zakładanej funkcjonalności rozpoczęto od analizy zbioru zestawu instrukcji SSE (ang. Streaming SIMD Extensions) umożliwiających wykorzystanie mechanizmu SIMD oraz napisania funkcji realizujących wszystkie zakładane operacje. Następnie przystąpiono do realizacji funkcji wykorzysującej koprocesor FPU do wykonywania sekwencyjnych operacji na pojedynczych liczbach.

```
// Initialising the struct of 128-bit vector
typedef struct{
    float vecOne;
    float vecTwo;
    float vecThree;
    float vecFour;
}numericVector;
```

Generowanie liczb

W celu przeprowadzenia pomiaru dla dużych zestawów danych liczby zmiennoprzecinkowe zostały wygenerowane z wykorzystaniem funkcji pseudolosowej rand() należącej do biblioteki stdlib.h. Otrzymane liczby losowany były w zakresie do miliona i dzielone przez inną wygenerowaną losowo liczbę, tak, aby większość z wygenerowanych liczb reprezentowała ułamek.

```
void generaetNumber(numericVector *dataSet){
    for(int i = 0; i<TAB_SIZE; ++i){
    dataSet[i].vecOne = (((float)(rand()%100000))/((float)(rand()%100)+1));
    dataSet[i].vecTwo = (((float)(rand()%100000))/((float)(rand()%100)+1));
    dataSet[i].vecThree = (((float)(rand()%100000))/((float)(rand()%100)+1));
    dataSet[i].vecFour = (((float)(rand()%100000))/((float)(rand()%100)+1));
    }
}
```

Pomiar czasu

Zarówno w przypadku operacji SIMD jak i SISD do pomiaru czasu wykorzystana została biblioteka time.h. Przed oraz po wykonaniu funkcji w assemblerze wywoływana była clock(), która zwracała ilość cykli procesora. Po pobraniu całkowitej ilości cykli procesora potrzebnej na wykonanie działania wartość dzielona była przez stałą CYCLES_PER_SEC, co pozwoliło na otrzymanie czasu trwania pojedynczej operacji w sekundach, po czym wartość była dodawana do sumarycznego czasu trwania dla danego zestawu danych i operacji. Czasy każdej próby były dodawane do sumarycznego czasu wykonywania danej operacji w tablicy times, w której komórki odpowiadały odpowiednio sumarycznym czasom operacji dodawania, odejmowania, mnożenia, dzielenia. Przy zapisie danych do pliku wynikowego czas dzielony jest przez całkowitą liczbę prób, tak aby otrzymać średni czas wykonania jednej operacji.

```
for(int j = 0; j < 3;++j){
    for(int i = 0; i < REPETITION; ++i){
    generaetNumber(setA);
    generaetNumber(setB);
        for(int k = 0; k<(numberOfNumbers[j]/4); ++k){
            //additionSIMD(&setA[k], &setB[k]);
            times[0] += (double)additionSIMD(&setA[k], &setB[k]);
            times[1] += (double)subtractionSIMD(&setA[k], &setB[k]);
            times[2] += (double)multiplicationSIMD(&setA[k], &setB[k]);
            times[3] += (double)divisionSIMD(&setA[k], &setB[k]);
        }

    }
```

```
for(int j = 0; j < 3;++j){
    for(int i = 0; i < REPETITION; ++i){
    generaetNumber(setA);
    generaetNumber(setB);
        for(int k = 0; k<(numberOfNumbers[j]/4); ++k){
            times[0] += (double)additionSISD(&setA[k].vecOne, &setB[k].vecOne);
            times[1] += (double)subtractionSISD(&setA[k].vecOne, &setB[k].vecOne);
            times[2] += (double)multiplicationSISD(&setA[k].vecOne, &setB[k].vecOne);
            times[3] += (double)divisionSISD(&setA[k].vecOne, &setB[k].vecOne);
        }
        for(int k = 0; k<(numberOfNumbers[j]/4); ++k){
            times[0] += (double)additionSISD(&setA[k].vecTwo, &setB[k].vecTwo);
            times[1] += (double)subtractionSISD(&setA[k].vecTwo, &setB[k].vecTwo);
            times[2] += (double)multiplicationSISD(&setA[k].vecTwo, &setB[k].vecTwo);
            times[3] += (double)divisionSISD(&setA[k].vecTwo, &setB[k].vecTwo);
        }
        for(int k = 0; k<(numberOfNumbers[j]/4); ++k){
            times[0] += (double)additionSISD(&setA[k].vecThree, &setB[k].vecThree);
            times[1] += (double)subtractionSISD(&setA[k].vecThree, &setB[k].vecThree);
            times[2] += (double)multiplicationSISD(&setA[k].vecThree, &setB[k].vecThree);
            times[3] += (double)divisionSISD(&setA[k].vecThree, &setB[k].vecThree);
```

```
        }
        for(int k = 0; k<(numberOfNumbers[j]/4); ++k){
            times[0] += (double)additionSISD(&setA[k].vecFour, &setB[k].vecFour);
            times[1] += (double)subtractionSISD(&setA[k].vecFour, &setB[k].vecFour);
            times[2] += (double)multiplicationSISD(&setA[k].vecFour, &setB[k].vecFour);
            times[3] += (double)divisionSISD(&setA[k].vecFour, &setB[k].vecFour);
            divisionSISDResult(&setA[k].vecFour, &setB[k].vecFour);
        }
    }
```

Mierzenie czasu obliczeń arytmetycznych – SISD

Implementacja operacji arytmetycznych opartych o mechanizm SISD odbywa się w oparciu o zestaw instrukcji FPU. Każda operacja wykonywana jest w osobnej funkcji, do której przekazywane są dwie liczby zmiennoprzecinkowe o pojedynczej precyzji. Przed wykonaniem wstawki napisanej w assmeblerze pobierana jest aktualna ilość cykli procesora. Za pomocą mnemonika fld ładowany jest pierwszy argument, który trafia na stos ST. Następnie za pomocą instrukcji ze zbioru {fadd, fsub, fmul, fdiv} wykonywana jest operacja z wykorzystaniem drugiego argumentu przekazywanego w funkcji. Wymuszenie „=m” wykorzystywane jest, by powiadomić kompilator o tym, że operand jest zmienionym adresem pamięci. Wynik poszczególnych działań zapisywany jest w zmiennej result. Dzięki ograniczeniu „-t” zapisanym przy zmiennej result, wartość po wykonaniu działania zdejmowana jest ze stosu ST. Po wykonaniu danej operacji ponownie pobierana była ilość cykli procesora. Funkcja mierząca czas zwracała różnicę w ilości cykli procesora między rozpoczęciem wykonania działania operacji a zakończeniem podzielonym przez stałą ilość cykli w sekundzie – CYCLES_PER_SEC, w wyniku czego zwracała czas potrzebny do wykonania działania (podany w sekundach).

```
double divisionSISD(float *a, float *b){
float result;
timeOfStart_t = clock();

asm(
    "fld %1\n"
    "fdiv %2 \n"
    :"=t"(result)
    :"m"(*a), "m"(*b)
);
timeOfEnd_t = clock();
timePassed= (double)(timeOfEnd_t - timeOfStart_t)/CLOCKS_PER_SEC;
return timePassed;
}
```

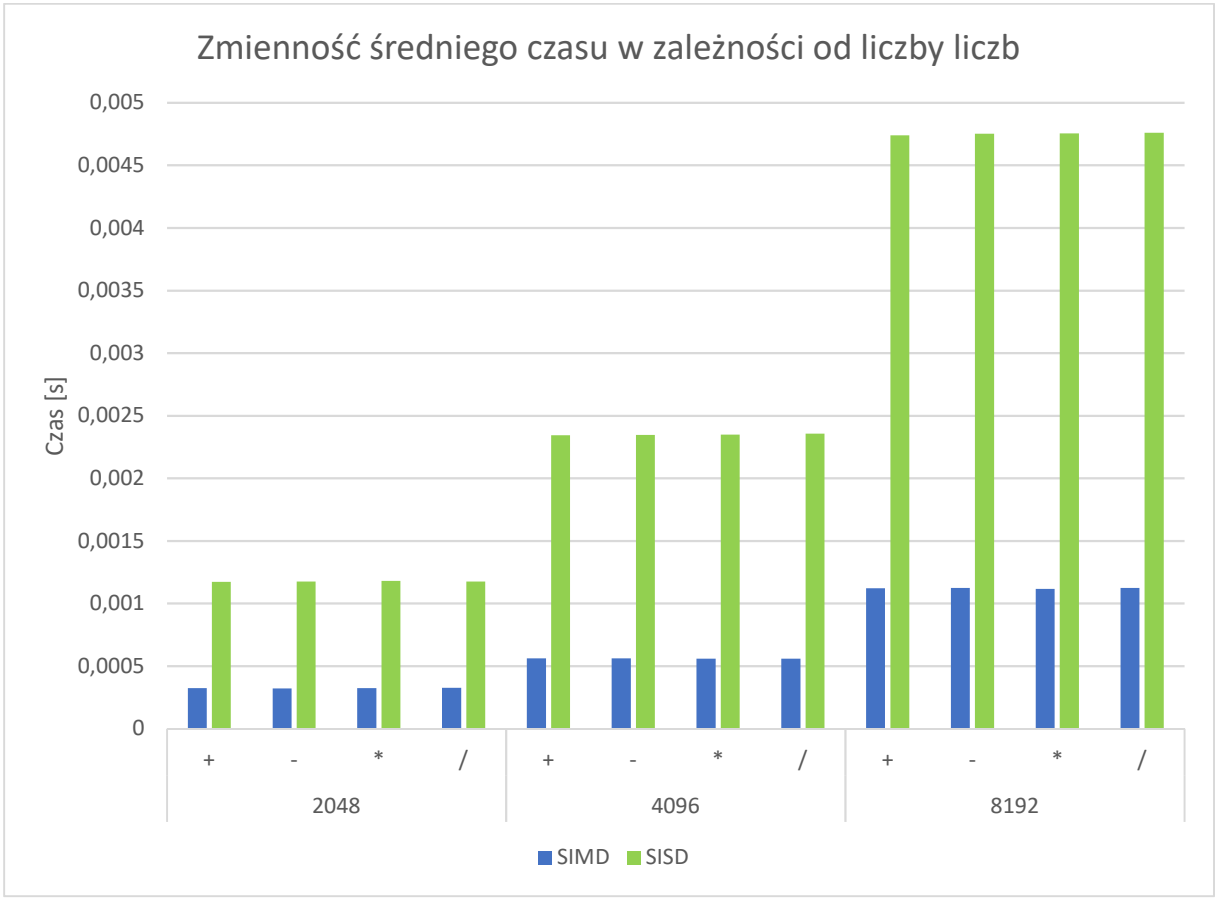
Mierzenie czasu obliczeń arytmetycznych – SIMD

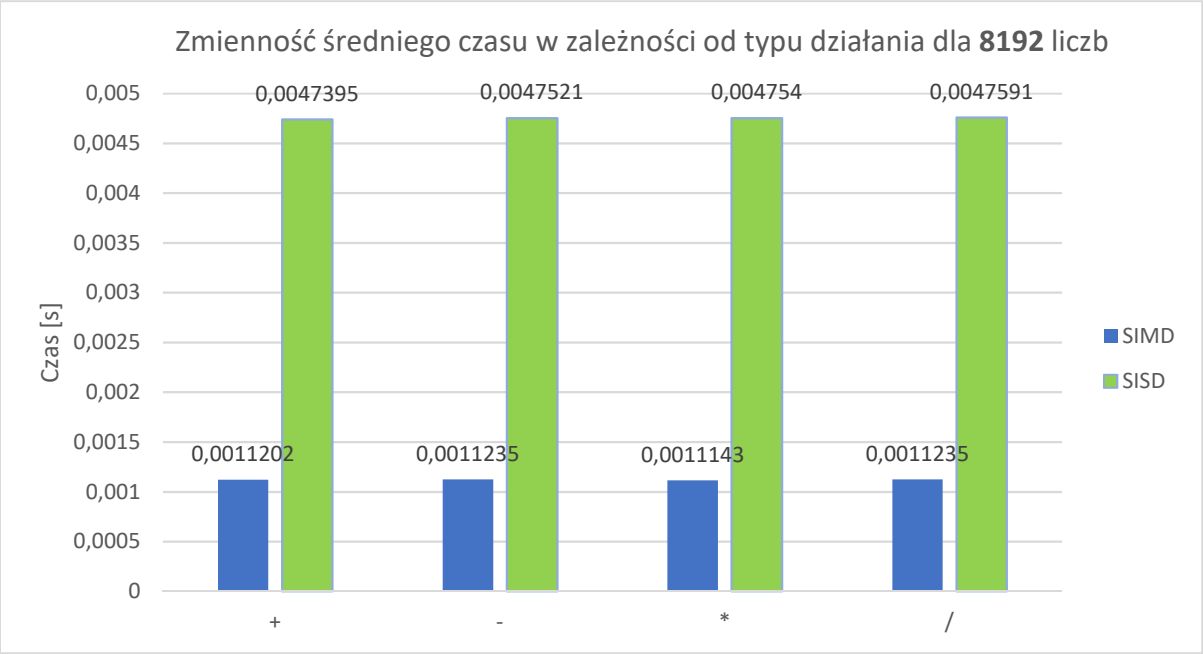
Operacje na wektorach 128 bitowych zostały wdrożone przy pomocy zbioru funkcji SSE. Po pobraniu ilości cykli procesora przed rozpoczęciem wykonywania działań, 128 – bitowy wektor przechowujący pierwszy zestaw liczb zostaje załadowany do rejestru XMM0 z pomocą rejestru RAX (zapisany jest jako argument pierwszy). Następnie drugi z przekazywanych wektorów zostaje załadowany do rejestru XMM1 (argument wejścia numer dwa). Na danych w rejestrach XMM, w których zapisane są dane zostają wykonane instrukcje addps, subps, mulps oraz divps w zależności od pomiaru poszczególnych operacji. Następnie wynik przypisywany jest zerowemu operandowi (result). Jako ostatni argument wpisywany jest CLOBBERED REGISTER. Po zakończeniu wykonywania działania wstawki ponownie sczytywana jest ilość cykli procesora a następnie analogicznie do funkcji SISD zwracany jest czas trwania funkcji.

```
double divisionSIMD(numericVector* setA ,numericVector *setB){
    numericVector result;

    timeOfStart_t = clock();

    asm(
        "movq %1, %%rax\n"
        "movups (%%rax), %%xmm0\n"
        "movq %2, %%rax\n"
        "movups (%%rax), %%xmm1\n"
        "divps %%xmm1, %%xmm0\n"
        "movups %%xmm0, %0\n"
        :"=m"(result)
        :"m"(setA), "m"(setB)
        :"rax"
    );

    timeOfEnd_t = clock();
    timePassed= (double)(timeOfEnd_t - timeOfStart_t)/CLOCKS_PER_SEC;

    return timePassed;
}
```

Otrzymane wyniki oraz wykresy

ILOSC LICZB	OPERACJA	SIMD	SISD
2048	DODAWANIE	0.000323	0.001172
	ODEJMOWANIE	0.00032	0.001174
	MNOŻENIE	0.000322	0.00118
	DZIELENIE	0.000325	0.001174
4096	DODAWANIE	0.000561	0.002343
	ODEJMOWANIE	0.00056	0.002346
	MNOŻENIE	0.000559	0.002349
	DZIELENIE	0.000558	0.002356
8192	DODAWANIE	0.00112	0.00474
	ODEJMOWANIE	0.001124	0.004752
	MNOŻENIE	0.001114	0.004754
	DZIELENIE	0.001124	0.004759





Podsumowanie

Na podstawie przedstawionych wykresów widać, iż działania wykorzystujące mechanizm SIMD wykonane zostały w około 4 razy mniejszym czasie niż operacje zaimplementowane przy pomocy SISD . Średni zysk procentowy czasu trwania operacji wynikający z zastosowania mechanizmu SIMD zamiast SISD wyniósł około **75,05%.** Został on obliczony na podstawie równania :

$$Zysk\ procentowy = \frac{\acute{S}redniCzasSISD - \acute{S}redniCzasSIMD}{\acute{S}redniCzasSISD} \cdot 100\%$$

Analizując drugi z wykresów można również zwrócić uwagę, iż mimo dużej ilość danych różnice pomiędzy czasami poszczególnych działań dla tej samej realizacji przetwarzania, nie różnią się znacząco.

Plik uruchomieniowy

Poniżej przedstawiona została zawartość pliku uruchomieniowego makefile.

```
all:
	gcc lab4.c -o lab4
```

Napotkane problemy i wnioski

Realizacja zadań laboratoryjnych przebiegła bez większych problemów. Najtrudniejszym zagadnieniem był przepływ danych i wymiana danych pomiędzy funkcją w języku C, a wstawkami z assemblera. Ponadto w celu sprawdzenia poprawności obliczeń napisane zostały funkcje dodatkowe sprawdzające wynik poszczególnych operacji z wykorzystaniem SSE oraz FPU, początkowo wyniki obliczane na z wykorzystaniem mechanizmu SISD, były w większości równe zero. Spowodowane to było faktem błędnego przekazania argumentów do wstawki w assemblerze (brak asteriksa przy zmiennych wejściowych). Zlokalizowanie problemu okazało się bardzo czasochłonne. W trakcie realizacji zadnia poszerzona została wiedza na temat sposobów przetwarzania działań w nowoczesnych procesorach. Analizując otrzymane wyniki można zobaczyć, iż wykorzystanie architektury SSE pozwala znacząco przyspieszyć wykonywanie obliczeń. Z niewiadomych powodów dla pierwszych prób wyniki pomiarów czasu działań na 2048 liczbach w schemacie SIMD były mocno zawyżone, zbliżone były do wyników pomiarów dla 4096 liczb. Wraz z kolejnymi próbami czas potrzebny na wykonanie 2048 operacji zbliżył się do zakładanego. Możliwe, iż było to spowodowane niewielką ilością powtórzeń testów (10 powtórzeń) oraz okresowo zwiększonym obciążeniem procesora w trakcie wykonywania pomiarów.