Politechnika Wrocławska

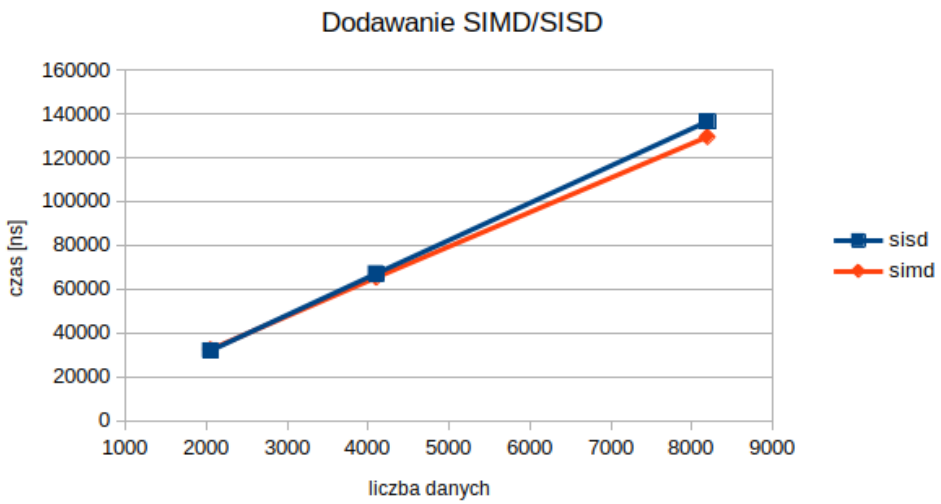
Sprawozdanie z laboratorium nr 3
Mechanizmy SISD I SIMD

Łukasz Wdowiak

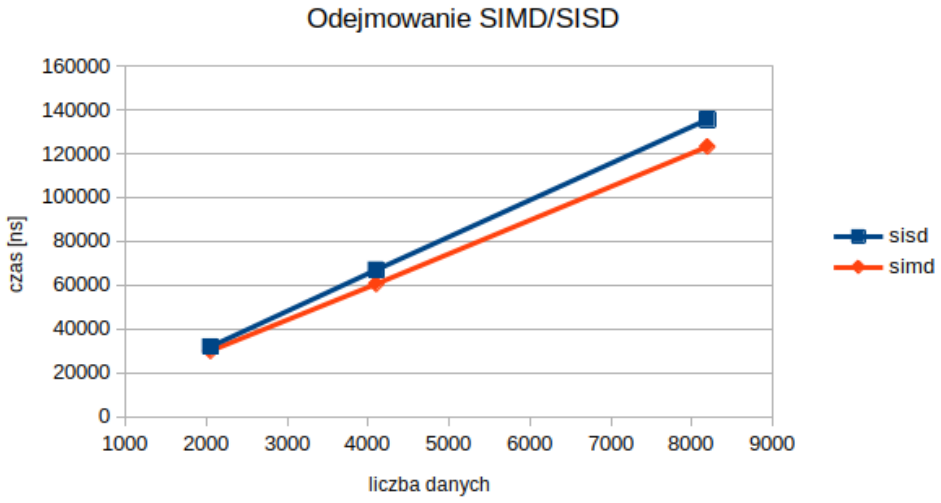
Prowadzacy: mgr inż. Tomasz Serafin

Wydział Informatyki i Telekomunikacji
Informatyka Techniczna
IV semestr

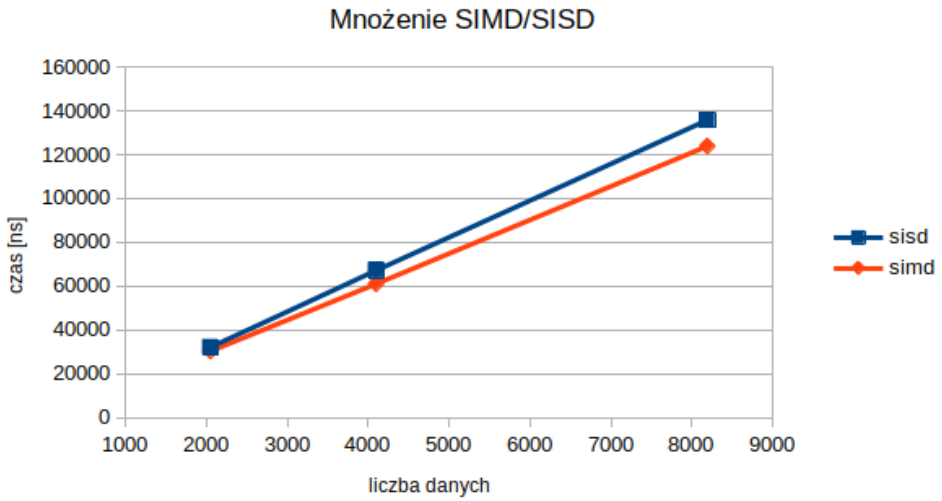
1 Wykresy



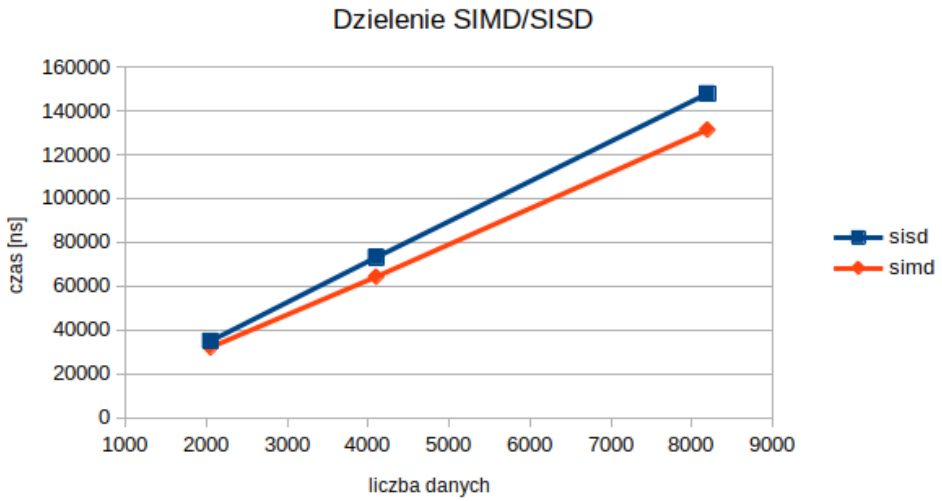
Rysunek 1: wykres średniego czasu dla dodawania



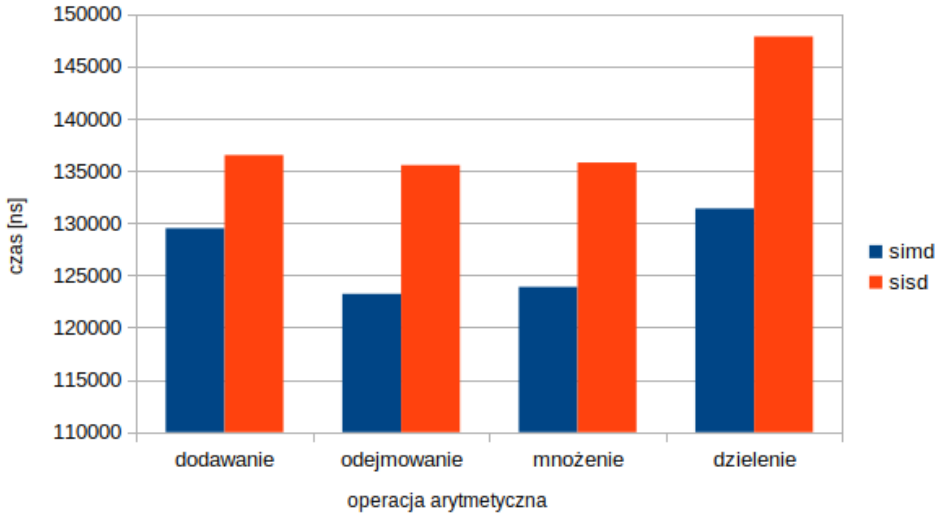
Rysunek 2: wykres średniego czasu dla odejmowanie



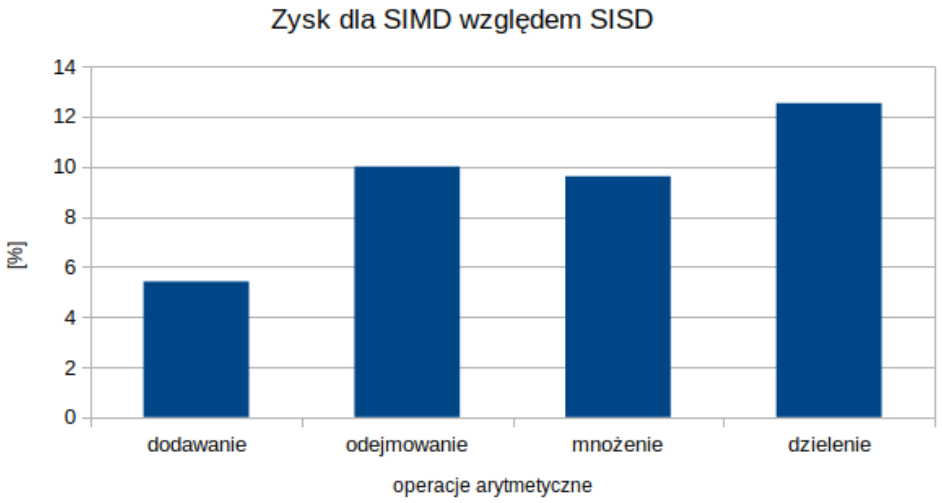
Rysunek 3: wykres średniego czasu dla mnożenia



Rysunek 4: wykres średniego czasu dla dzielenia



Rysunek 5: wykres średniego czasu od typu działania dla 8192 liczb



Rysunek 6: wykres zysku dla SIMD względem SISD dla 8192 liczb

2 Przebieg pracy nad programem

1. Stworzenie makefile
2. Stworzenie funkcji odpowiedzialnej za zapis do pliku
3. Research na temat SIMD
4. Stworzenie funkcji zapełniajacej wektory losowymi liczbami
5. Stworzenie funkcji dodawania/odejmowania/mnożenia/dzielenia dla SIMD
6. Research na temat SISD
7. Stworzenie funkcji zapełniajacej tablice losowymi liczbami
8. Stworzenie funkcji dodawania/odejmowania/mnożenia/dzielenia dla SISD
9. Napisanie funkcji testujacych
10. Urozmaicenie programu o parametry startowe (liczba powtórzeń, rozmiar tablicy)

3 Napotkane problemy

- zapis składni assemblerowej w c++
- zbyt losowe wyniki pomiarów przy małej ilości testów

4 Kluczowe fragmenty kodu

Przykładowa funkcja dodawania dla SIMD:

```
double addVectorSIMD(Vector *v1, Vector *v2, Vector *result)
{
    auto start = std::chrono::high_resolution_clock::now();
    __asm__ __volatile__(
        "movups (%0), %%xmm0\n"
        "movups (%1), %%xmm1\n"
        "addps %%xmm1, %%xmm0\n"
        "movups %%xmm0, (%2)\n"
        :
        : "r"(v1), "r"(v2), "r"(result)
        : "%xmm0", "%xmm1");

    auto end = std::chrono::high_resolution_clock::now();
    return std::chrono::duration_cast<std::chrono::nanoseconds>(end - start).count
}
```

Czas wykonania operacji matematycznej jest mierzony za pomocą biblioteki chrono. Wstawka assemblerowa znajduje się w instrukcji

```
__asm__ __volatile__ ();
```

co oznacza mniej więcej tyle, że kod który teraz napiszemy będzie w assemblerze, a kompilator ma go nie optymalizować. Kod assemblerowy przenosi wartość z argumentu pierwszego i drugiego do odpowiednio xmm0 i xmm1. Następnie użyta jest instrukcja dodania dwóch wektorów. Na sam koniec z xmm0 przenosimy wynik naszego dodawania do trzeciego argumentu naszej funkcji, czyli zmiennej result. Pierwszy dwukropek jest wymagany jako dane wejściowe, drugi za dane wejściowe do kodu assemblera, trzecie za dane do zniszczenia.

5 Opis uruchomienia programu

Kompilacja programu:

```
g++ -std=c++11 -o main main.cpp
```

Program urachamiany jest z pomocą makefile. W celu uruchomienia programu należy wpisać w konsoli "make", a następnie "./main". W celu uruchomienia programu z parametrami należy wpisać "./main [liczba powtórzeń] [rozmiar tablicy]".

6 Czym jest SIMD i SISD - rys historyczny

6.1 SISD (Single Instruction, Single Data)

SISD to tradycyjny model obliczeń, w którym pojedyncza instrukcja jest wykonywana na pojedynczych danych. W takim przypadku procesor wykonuje kolejne instrukcje sekwencyjnie, przetwarzając jedną wartość danych na raz. To jest typowa architektura używana w większości konwencjonalnych procesorów.

6.2 SIMD (Single Instruction, Multiple Data)

SIMD to architektura, w której jedna instrukcja jest wykonywana jednocześnie na wielu zestawach danych. Oznacza to, że można równocześnie przetwarzać wiele danych przy użyciu jednej instrukcji, co może prowadzić do zwiększenia wydajności obliczeń. Procesory SIMD są zdolne do równoczesnego przetwarzania dużej liczby operacji, szczególnie przy obliczeniach równoległych, takich jak operacje na macierzach, przetwarzanie obrazów czy symulacje fizyczne.

6.3 Rys historyczny

SISD był dominującym modelem obliczeń na początku ery komputerów, gdzie jedna instrukcja była wykonywana sekwencyjnie na pojedynczych danych. W latach 60. i 70. XX wieku powstała architektura SIMD, umożliwiająca równoczesne przetwarzanie wielu danych. SIMD zyskało popularność w latach 80. i 90. dzięki rozwojowi technologii multimedialnych i potrzeb przetwarzania danych wideo i dźwięku w czasie rzeczywistym. .

7 Wnioski

Z wykresów jasno wynika, że SIMD jest zdecydowanie szybsze. Zależność czasu wykonywania działań od liczby danych jest liniowa. Przy małej ilości testów tych samych wektorów (10 testów) niestety trudno zauważyć panujący schemat w różnicy pomiędzy SIMD, a SISD, gdyż czasy wykonywania operacji matematycznych mogą być różne, czasem wręcz losowe, z racji tego, że nie wiemy jakie operacje przeprowadza w tym momencie nasz system operacyjny. Dlatego też zwiększyłem liczby testów do 1000, co pozwala uzyskać bardziej wiarygodne wyniki. Średni zysk SIMD wzgledem SISD dla róznych operacji wynosi około 10%. SIMD zdecydowanie przyspiesza wykonywanie trudnych operacji matematycznych takich jak dzielenie.