# lab_12 - Instrukcja do ćwiczenia

**Teoria:**

http://galaxy.agh.edu.pl/~amrozek/AK/lab12.pdf

**Transpozycja macierzy:**

Transpozycja macierzy (zamiana wierszy z kolumnami) pozwala na uproszczenie odwołań do danych w trakcie liczenia iloczynu macierzy – w klasycznym podejściu, najbardziej wewnętrzna pętla (i najczęściej wykonywana) zawiera odwołania do kolumn macierzy **A** i wierszy macierzy **B**. Ponieważ dwuwymiarowa tablica (macierz) zapisana jest w pamięci wiersz po wierszu, to dostęp do niej jest prosty i szybki o ile odwołania dotyczą tego samego wiersza (cały wiersz może znaleźć się w pamięci cache). Oznacza to prosty dostęp do macierzy **B** i skomplikowany (czasochłonny) do macierzy **A**. Gdyby w najbardziej wewnętrznej pętli, odwoływać się zarówno do wierszy macierzy **B**, jak i wierszy macierzy **A**, to powinno to wpłynąć na uproszczenie odwołań i przyspieszenie działania. Zmiana sposobu odwołań wiąże się jednak z koniecznością poprzestawiania danych w macierzy **A** (jej transpozycji), tak aby uzyskany rezultat był faktycznie iloczynem macierzy.

Transpozycję macierzy można zrealizować przy pomocy następującego kodu:

```
void transpose( int n, double *X, double *Y )
{
 register int i, j;
 register double val;

 for(i=0;i<n;i++)
   for(j=0;j<n;j++)
     Y[i*n+j] = X[i+j*n];
}
```

Użycie transponowanych danych wiąże się z koniecznością modyfikacji kodu wszystkich wykorzystywanych metod liczenia iloczynu macierzy – przykładowo: odpowiednikiem funkcji **dgemm_naive** jest funkcja **dgemm_naive_trans**, itp. Aby porównanie czasów działania poszczególnych metod było miarodajne, to pomiar czasu dla funkcji **xxxxx_trans** powinien też uwzględniać realizację samej transpozycji.

**Praktyka (lab_12.c):**

**Działania:**

1. Przygotowujemy kod do mierzenia czasów wykonania zawarty w **eval_time.c**.
2. C (Compile) – polecenie: **gcc –O3 –c eval_time.c**
3. Przechodzimy do programu **lab_12.c** – jest to zmodyfikowana wersja programu **mat_mat.c**, a wprowadzone zmiany dotyczą wykorzystania algorytmów wykorzystujących dane po transpozycji oraz możliwości dokonania wszystkich pomiarów w trakcie pojedynczego uruchomienia programu.
4. CL (Compile, Link) – polecenie: **gcc –O3 –o lab_12 lab_12.c eval_time.o**
5. R (Run) – polecenie: **./lab_12**
6. Przykładowe efekty uzyskane po uruchomieniu wyglądają następująco:

. . .

7. Widać, że metody wykorzystujące transpozycję działają szybciej niż ich klasyczne odpowiedniki.
8. Uruchamiamy program kilkukrotnie (im więcej tym lepiej w kontekście wiarygodności i powtarzalności) – aby ułatwić sobie dalsze prace można wyniki zapisywać do plików tekstowych:

```
./lab_12   >  result_1.txt
./lab_12   >  result_2.txt
      …
./lab_12   >  result_n.txt
```

9. Wykorzystujemy zgromadzone dane (po odrzuceniu tych, które znacznie się różnią od pozostałych) do szczegółowej analizy i prezentacji w formie tabeli/wykresu.
10. Analiza wygenerowanego przez kompilator kodu w asemblerze (po użyciu polecenia **gcc –O3 –S lab_12.c** uzyskamy plik **lab_12.s**) pozwala na porównanie metod przy użyciu innych kryteriów niż tylko szybkość działania – mogą to być: długość kodu porównywalnych funkcji (liczba instrukcji w funkcjach **xxxx** i **xxxx_trans**), liczba użytych rejestrów **xmmk**, liczba odwołań do danych umieszczonych na stosie, itp.