# Programowanie Równoległe na Architekturach Wielordzeniowych

# Parallel programming for multi-core architectures

# 2023/2024

# Parallel implementation evaluation report

1. Project details

|  |  |
| --- | --- |
| Project title: | Matix Multiplication |
| Student’s name: | Łukasz Smoliński |
| Index number: | 184306 |
| Email: | s184306@student.pg.edu.pl |

1. Description of data used for experiments (including examples, when possible)

|  |
| --- |
| W przeprowadzonych eksperymentach wykorzystano dane w postaci macierzy o różnorodnych wymiarach, które zawierały zarówno losowe wartości, jak i wartości stałe (fixed). W przypadku losowych danych elementy macierzy były generowane jako liczby zmiennoprzecinkowe z przedziału [0, 1]. Dla danych stałych wszystkie elementy macierzy przyjmowały tę samą wartość np. równą 2. Macierze były generowane na nowo przed każdym uruchomieniem testu. Generacja dany zapewniała spełnienie warunku mnożenia macierzy.   Przy teście wydajności wykorzystano dla CUDA-based macierze o losowych elementach o rozmiarach: A: 30 000 x 30 000  B: 30 000 x 30 000  W OpenMP-based:  A: 2500 x 2500  B: 2500 x 2500 |

1. Environment #1 description

|  |
| --- |
| Operating system: Distributor ID: Ubuntu  Description: Ubuntu 22.04.5 LTS  Release: 22.04  Codename: jammy  CPU:  13th Gen Intel(R) Core(TM) i7-13700K  Architektura: x86\_64  GPU:  NVIDIA GeForce RTX 4070 Ti  CUDA: nvcc: NVIDIA (R) Cuda compiler driver  Copyright (c) 2005-2022 NVIDIA Corporation  Built on Wed\_Sep\_21\_10:33:58\_PDT\_2022  Cuda compilation tools, release 11.8, V11.8.89  Build cuda\_11.8.r11.8/compiler.31833905\_0 |

1. Environment #2 description

|  |
| --- |
| *OpenMP version: 201511*  *Maksymalna liczba wątków: 24*  *Liczba wykorzystanych wątków: 8*  *Reszta tak samo jak wcześniej* |

1. Test results

|  |  |  |
| --- | --- | --- |
| **Implementation** | **Execution time\*** | |
| **Mean [s]** | **Uncertainty [s]** |
| CUDA-based | 27.041 s | 0.149 s |
| OpenMP-based | 5,873 s | 0,192 s |

\* calculated over 10 executions, uncertainty calculated as:

1. Implementation #1 details

|  |
| --- |
| W ramach projektu przygotowano dwie wersje implementacji mnożenia macierzy na GPU przy użyciu CUDA:   * Wersja pierwsza (v1) – Podstawowa implementacja, w której każdy wątek oblicza jeden element wynikowej macierzy. Dane były bezpośrednio odczytywane z pamięci globalnej GPU. Ta wersja jest prostsza w implementacji, ale charakteryzuje się większymi opóźnieniami związanymi z częstym dostępem do pamięci globalnej. * Wersja druga (v2) – Zoptymalizowana implementacja, wykorzystująca pamięć współdzieloną (shared memory) do przechowywania fragmentów macierzy wejściowych. Dzięki temu ograniczono liczbę operacji odczytu z pamięci globalnej i poprawiono wydajność obliczeń. Każdy blok wątków oblicza sub-macierz wynikowej macierzy, synchronizując wątki w celu efektywnego zarządzania danymi w pamięci współdzielonej.   Wersja v2, dzięki zastosowaniu pamięci współdzielonej, okazała się znacznie bardziej wydajna w porównaniu do wersji v1, szczególnie dla większych macierzy. Redukcja liczby operacji dostępu do pamięci globalnej miała kluczowy wpływ na skrócenie czasu wykonania.  Wynik testu dla v1: 32.297 s  Wynik testu dla v2: 27.041 s  W ostatecznej implementacji zastosowano także mechanizm paddingu, umożliwiający mnożenie macierzy o dowolnych wymiarach, niekoniecznie będących wielokrotnością rozmiaru bloku (BLOCK\_SIZE). Wyniki były ostatecznie kopiowane i prezentowane w oryginalnym rozmiarze macierzy, co zapewniało zgodność obliczeń z oczekiwaniami. |

1. Implementation #2 details

|  |
| --- |
| Przygotowano dwie wersje implementacji mnożenia macierzy na CPU z użyciem OpenMP:   * **Wersja pierwsza (v1)** – Podstawowa implementacja:   + Reprezentacja macierzy jako jednowymiarowej tablicy.   + Każdy wątek bierze dany element i, j macierzy C (wynikowej) i oblicza jego wartość – czyli wykonuje mnożenie skalarne i-tego wiersza A i j-tej kolumny B   + Równoległość na poziomie iteracji po wierszach i kolumnach (#pragma omp parallel for collapse(2)).   + Wykorzystano schedule static * **Wersja druga (v2)** – Zoptymalizowana implementacja:   + Zmiana reprezentacji macierzy na dwuwymiarową tablicę (float\*\*), poprawiając lokalność pamięci.   + Dodanie wektoryzacji (#pragma omp simd) w pętli obliczającej dany element macierzy C (wynikowej).   + Redukcja false sharing poprzez ograniczenie dostępu do współdzielonej macierzy wynikowej.   Wynik testu dla v1: 11,101 s  Wynik testu dla v2: 5,873 s |

1. Survey

Fill the answers for questions related to frameworks that you used in your project:

* 1. How many lines of code did you write for:
     1. OpenMP implementation: 55
     2. CUDA/OpenCL implementation: 122
  2. How would you describe programming difficulty of each framework/interface in 1-10 scale (1 – easy, 10 – difficult):
     1. OpenMP: 3
     2. CUDA/OpenCL: 6