Wrocław 2020.12.11
Autorzy: Jerzy Sas, Michał Przewoźniczek

Trzeci Konkurs Programowania Obiektowego
Czyli na co komu C++ i obiekty?
Edycja 2020 – Optymalizacja operacji macierzowych

I. Cel i warunki konkursu

Celem konkursu jest dobra zabawa w rozwiązywanie łamigłówek programistycznych, które jednocześnie są zadaniem otwartym. Celem zadania konkursowego jest zaprojektowanie i implementacja kompletnego efektywnego czasowo zestawu komponentów wspierających obliczenia macierzowe, tak aby zaimplementowane zostały operatory pozwalające stosować notację zbliżoną do używanej w algebrze liniowej.

Należy zaprojektować i zaimplementować klasę reprezentującą macierz (w szczególności macierz o wymiarach 1xN i Nx1 odpowiadającą wektorowi wierszowemu i kolumnowemu). W klasie tej należy zaimplementować operatory pozwalające w naturalny i zbliżony do przyjętej notacji algebraicznej zapisywać wyrażenia opisujące operacje na macierzach. W szczególności należy zaimplementować operatory do:

- dodawania i odejmowania macierzy
- mnożenia macierzy (w tym mnożenia wektorów i macierzy),
- mnożenia macierzy przez liczbę,
- transponowania macierzy.

Dodatkowo należy zaimplementować operacje pozwalające na:

- tworzenie macierzy o zadanych wymiarach (w konstruktorze),
- przypisywanie i odczyt wartości wskazanych elementów macierzy,
- tworzenie wektorów poprzez wybór wskazanego wiersza lub kolumny macierzy,
- przypisywania macierzy kwadratowej wartości odpowiadających macierzy jednostkowej.

Wszystkie operacje (z wyjątkiem tych wykonywanych w naturalny sposób w konstruktorach) należy zaimplementować jako odpowiednio przeciążone operatory. Klasę należy zaimplementować jako **klasę dla typu float** (w odróżnieniu od zadania laboratoryjnego nie jest klasa implementująca macierz nie musi być szablonowa). **Całość należy zaimplementować samodzielnie.**

W implementacji szczególną uwagę należy zwrócić na efektywność czasową i pamięciową swojej implementacji oraz zapewnić właściwą obsługę błędów. Implementacja nie może nakładać żadnych jawnych ograniczeń na rozmiary przetwarzanych macierzy. W szczególności należy zapewnić aby możliwe było przetwarzania macierzy o wymiarach rzędu 1000x1000 i większych.

Studenci mogą ze swoimi implementacjami operacji macierzowych przystąpić do konkursu na najefektywniejszą implementację. Kryterium konkursu jest czas wykonania testowych obliczeń z wykorzystaniem metod i funkcji operatorowych zaimplementowanych przez uczestnika konkursu. Zasady konkursu są następujące:

- Czas wykonania obliczeń testowych z wykorzystaniem dostarczonego kodu będzie porównany z czasem identycznych obliczeń wykonanych z wykorzystaniem implementacji referencyjnej opartej o niezoptymalizowane implementacje oparte bezpośrednio na definicjach poszczególnych operacji. **Do konkursu zostaną zakwalifikowane wyłącznie, które będą co najmniej 3 razy szybsze od implementacji referencyjnej.**
- Kryterium oceny będzie współczynnik przyspieszenia względem implementacji referencyjnej liczony jako stosunek

$$Q = \frac{t_{ref}}{t_{prog}}$$

  gdzie t_{ref} to czas obliczeń implementacji referencyjnej a t_{prog} to czas obliczeń implementacji konkursowej.
- Kryterium dopuszczającym do dalszych etapów konkursu jest osiągnięcie współczynnika przyspieszenia na poziomie co najmniej $Q=3.0$. Oceniana będzie tylko efektywność czasowa.
- Nie jest oceniana efektywność pamięciowa (tzn. w celu poprawienia efektywności czasowej można stosować dowolne struktury danych bez względu na zajętość pamięci).
- Ograniczona jest jedynie rozmiar całkowitej pamięci alokowanej przez program. Przy utworzeniu 10 macierzy o wymiarach 10000x10000 pamięć alokowana przez program, który wykonuje dowolne obliczenia na tych macierzach nie powinna być większa niż 10GB.
- Wyniki obliczeń programu konkursowego będą porównane z wynikami obliczeń implementacji referencyjnej. Różnica dowolnej wartości skalarnej należącej do macierzy i wektorów wynikowych powyżej 1.0e-6 dyskwalifikuje zawodnika.
- Procedura konkursowa będzie wykonywana na komputerze z Windows 7 i CPU i7-5930K 3.5GHz.
- Zawodnik startujący w konkursie zobowiązany jest dostarczyć kod źródłowy programu oraz jego wersję skompilowaną, zdolną do wykonania w środowisku wyspecyfikowanym w poprzednim punkcie.
- Niespełnienie wymagań sprecyzowanych w pkt. 2 (w szczególności – niezrozumienie kodu przedstawionego do konkursu) skutkuje bezwarunkową dyskwalifikacją zawodnika.

Przykładowy kod źródłowy obliczeń wykorzystywanych do oceny czasu wykonania programów konkursowych będzie dostarczony zawodnikom. Jednak ostateczna postać procedury obliczeniowej użytej w konkursie będzie inna.

1. Wymagania dla kodu

1.1. **Język programowania.** Dozwolone jest wyłącznie użycie C++ w wersji **C++98 lub wyższej**. Dostarczony kod musi być zgodny z wymaganiami laboratorium TEP.

1.2. **Pozostałe wymagania.**

- nie można wykorzystywać żadnych obcych komponentów poza standardowym szablonami implementującymi kontenery, w szczególności nie można korzystać z bibliotek LinAlg, MKL, Blas Lapack i innych
- kod powinien być kompilowany w środowisku MSVC 2017 lub 2019,
- nie można wykorzystywać wspomagania przez GPU,
- w celu poprawy efektywności implementacji można stosować instrukcje SIMD (w wersja AVX i AVX2, nie dopuszczamy AVX-512 ze względu na ograniczoną dostępność w starszych procesorach),
- zabronione jest wykorzystywanie kodu pozyskanego metodą copy-and-paste,
- można stosować dowolne algorytmy wykonywania operacji macierzowych pod warunkiem, że ich implementacja jest w pełni dziełem autora oraz autor wykaże pełne zrozumienie wykorzystanej metody (tzn. nie można wykorzystywać obliczeń niezrozumiałych dla programisty),
- można wykorzystywać wielowątkowość przy ograniczeniu maksymalnej liczby wątków do 4.

1.3. Proponowany podział implementacji na etapy

Etap 1:

- opracowanie interfejsu klasy (metody publiczne, metody operatorowe, wybór algorytmów efektywnego mnożenia macierzy)
- implementacja prostych metod: dostęp do elementów macierzy, dodawanie, transpozycja, konstruktory i destruktory, mnożenie metodą "naiwną" (z definicji)

Etap 2:

- implementacja wybranej/wybranych metod mnożenia macierzy (można eksperymentować z wieloma metodami aby wybrać najefektywniejszą),
- optymalizacja kodu wykonującego mnożenie macierzy

Etap 3:

- implementacja zoptymalizowanych metod mnożenia macierzy (np. z wykorzystaniem instrukcji AVX, z uwzględnieniem ograniczenia programu przez dostęp do pamięci (optymalizacja wykorzystania pamięci podręcznej itp.)

2. Przypadki testowe

1.1. **Liczba przypadków testowych.** Wyniesie od 5 do 50 przypadków.
1.2. **Liczba przebiegów i jakość rozwiązania.** Każda implementacja zostanie uruchomiona 5 razy (ta liczba może ulec zmianie w zależności od liczby zgłoszeń) dla każdego przypadku testowego. Jako miara jakości implementacji dla danego problemu, zostaną przyjęte następujące wartości: najgorszy i najlepszy czas obliczeń oraz wartość średnia.
1.3. **Ranking.** Na podstawie tych wyników określonych zgodnie z miarami podanymi w poprzednim podpunkcie, zostanie utworzony ranking. Najlepsza implementacja otrzyma dla danego zadania 1 miejsce, druga 2 miejsce, itd. Jeśli więcej niż jedna implementacja będzie miała to samo miejsce to wszystkie one otrzymają to samo, najwyższe możliwe miejsce. Na przykład jest podany poniżej.

Mamy 3 implementacje (A,B i C) i 2 problemy (problem 1 i problem 2). Jako podstawową wartość klasyfikacji bierzemy pod uwagę średnią, potem wartość najlepszą, potem minimalną. Dla 2 problemów, wspomniane implementacje uzyskały następujące wyniki (im większa wartość tym lepiej). Przykładowe wyniki są przedstawione w tabeli.

	Problem 1			**Problem 2**		
Implementacja	**Średnia**	**Min**	**Max**	**Średnia**	**Min**	**Max**
Impl. A	1	1	1	0.5	0.4	0.7
Impl. B	1	0.8	1	0.6	0.3	0.6
Impl. C	1	1	1	0.5	0.4	0.7

Tabela 1. Przykładowe wyniki dla dwóch problemów i trzech metod.

Implementacja	**Problem 1**	**Problem 2**
Impl. A	1	2
Impl. B	3	1
Impl. C	1	2

Tabela 2. Przykładowy ranking dla wyników wskazanych w **Tabela 1**.

Komentarz. Dla problemu implementacje A i C osiągają najlepsze wyniki. Implementacja B ma tą samą średnią i tą samą wartość maksymalną, ale wartość minimalna jest gorsza niż dla implementacji A i C, dlatego implementacja B zajmuje dla Problemu 1, 3 miejsce, a implementacje A i C 1 miejsce. Dla problemu 2 sytuacja jest odmienna. Pierwsze miejsce zajmuje implementacja B, bo jej wyniki mają najwyższą średnią (to że minimum i maksimum są niższe niż dla implementacji A i C nie ma znaczenia). Implementacje B i C zajmują 2 miejsce. W ostatecznym rozrachunku, zwycięzcami są implementacje A i C (średnie miejsce: 1.5), a implementacja B zostaje uznana za najgorszą (średnie miejsce: 2).

II. Uczestnicy

W konkursie może wziąć udział każdy chętny, który zgłosi prawidłowo swój program. Nie trzeba nawet być studentem. Oczywiście nagroda numer 4, będzie przyznawana wyłącznie studentom, którzy obecnie uczęszczają na zajęcia TEP.

III. Techniczne przeprowadzenie konkursu i wyniki

1. Wymagany interfejs klasy, którą należy dostarczyć znajduje się w pliku *CMtx.h*. Przykładowy test znajduje się w pliku *Test_main.cpp*.

2. Żeby wziąć udział w konkursie należy do dnia 2021.01.18 do godziny 12:00 wysłać mail o tytule *Uwielbam C++, więc zgłaszam się do udziału w konkursie*, mail powinien zawierać w załączeniu pliki CMtx.h i CMtx.cpp.
   Mail zgłoszeniowy należy wysłać na dwa adresy:
   Jerzy.Sas@pwr.edu.pl
   michal.przewozniczek@pwr.edu.pl

3. Jedna osoba może zgłosić tylko jeden program. W przypadku wysłania programu kilka razy zostanie wzięta pod uwagę ostatnia prawidłowo przesłana wersja.

4. Komplet wyników wraz z odpowiednim raportem zostanie ogłoszony na ostatnim wykładzie w semestrze. Odpowiednie dane będą również dostępne na ePortalu.

IV. Nagrody

Główną nagrodą, dla każdego uczestnika konkursu jest **dobra zabawa** oraz **możliwość rozwiązania ciekawych i trudnych problemów za pomocą własnej metody obliczeniowej**. Ponadto, do zdobycia są następujące nagrody:

1. **Niepowtarzalny, oczywiście limitowany, okolicznościowy dyplom** z oryginalnym, odręcznym podpisem prowadzącego.

2. Pięciosekundowa pochwała wzrokowa udzielona na ostatnim wykładzie (warunkiem udzielenia tej nagrody jest obecność na tym wykładzie, prowadzący kategorycznie odmawia spoglądania w próżnię).

3. W przypadku chęci wzięcia udziału w wymianie studentów pomiędzy Politechniką Wrocławską, a **Missouri University of Science And Technology**, zdobywcy trzech pierwszych miejsc mogą liczyć na **list polecający od prowadzącego wykład**.

4. **Last but not least**. Zdobywcy 10 pierwszych miejsc mogą liczyć na **podniesienie oceny z laboratorium z TEP i wykładu z TEP o 0.5**. Uczestnikom z drugiej 10 o 0.5 podniesiona zostanie ocena z wykładu.

V. Komentarze i wskazówki

Najbardziej kosztownymi elementami obliczeń numerycznych są:
- wykonywanie zmiennoprzecinkowych operacji arytmetycznych, w szczególności mnożenia i dzielenia,
- dostęp do pamięci.

Ograniczeń efektywności związanej z wykonywaniem operacji arytmetycznych można unikać stosując następujące techniki:
Instrukcje maszynowe typu SIMD (Single Instruction, Multiple Data). Instrukcje te wykonują równocześnie tę samą operację (np. mnożenie) na całym wektorze operandów. W przypadku mnożenia macierzy gdzie każdy współczynnik macierzy wynikowej jest iloczynem skalarnym wiersza i kolumny wykorzystanie rozkazów tego typu może dać bardzo znaczące przyspieszenie. Warunkiem jest odpowiednie spakowanie wektorów stających się operandami rozkazu SIMD. Większość kompilatorów C++ umożliwia wplatanie w kod C++ rozkazów tego typu. Niektóre kompilatory analizują kod źródłowy i wykorzystują te instrukcje automatycznie. Zainteresowani tą techniką powinni zapoznać się z zasadami korzystania z instrukcji SIMD w dokumentacji używanego kompilatora.
Obliczenia wielowątkowe. Skomplikowane obliczenia można powierzyć niezależnie wykonującym się wątkom. Podstawowe warunki efektywnego wykonywania wątków to:
a) brak komunikacji i wewnętrznej synchronizacji pomiędzy wątkami
b) powierzenie wątkom stosunkowo długo trwających sekwencji obliczeń. Mechanizm uruchamiania, zatrzymywania i synchronizacji wątków jest bardzo kosztowny, stąd w przypadku krótkotrwałych wątków ostateczny efekt może być przeciwny do zamierzonego.

c) dostęp wątków do rozłącznych obszarów pamięci. Przy niespełnieniu tego warunku wątki mogą wzajemnie się wstrzymywać z powodu sterylizacji dostępu do pamięci.

Unikanie kosztownych operacji arytmetycznych. Należy pamiętać że wykonywanie operacji arytmetycznych na liczbach całkowitych jest znacznie szybsze niż wykonywanie tych operacji na liczbach zmiennoprzecinkowych. Tam gdzie możliwe należy stosować arytmetykę całkowitoliczbową. Należy również pamiętać, że w zakresie arytmetyki zmiennoprzecinkowej operacja mnożenia zajmuje więcej cykli zegarowych niż operacja dodawania a operacja dzielenie zajmuje (zwykle) więcej cykli niż operacja mnożenia. Należy w miarę możliwości unikać operacji dzielenia.

Dostęp do pamięci staje się czynnikiem ograniczającym efektywność wtedy gdy program w sekwencji następujących po sobie obliczeń wykorzystuje dane ulokowane w pamięci pod odległymi adresami. Obowiązuje zasada lokalności dostępu, mówiąca, że program (zwykle) działa efektywniej gdy w sekwencji kolejnych następujących po sobie obliczeń wykorzystuje się dane ulokowane w pamięci pod nieodległymi adresami. Gdy zasada ta nie jest zachowana nie jest w pełni wykorzystywana pamięć podręczna. Zasada jej działania polegająca na wczytywaniu do niej obszarów pamięci RAM całymi wierszami prowadzi wówczas nawet do zwiększenia czasu dostępu do danych (z pamięci), nawet w stosunku do architektury w której nie zrealizowano mechanizmu pamięci podręcznej. Należy pamiętać, że kompilator dodatkowo optymalizuje obliczenia tak aby unikać intensywnego dostępu do pamięci, np. wykorzystując przechowywanie wyników pośrednich w rejestrach procesora. Programista jednak musi "dać szansę" kompilatorowi na wykonanie takich optymalizacji odpowiednio aranżując w swoim programie kolejność wykonywania operacji i organizując właściwie struktury danych wykorzystywane w najbardziej intensywnych obliczeniach.

W przypadku operacji macierzowych ta przyczyna nieefektywności najwyraźniej występuje w przypadku wyliczania iloczynu skalarnego wiersza jednej macierzy i kolumny innej macierzy. O ile dostęp do kolejnych elementów wiersza zachowuje zasadę lokalności i nie zakłóca pracy pamięci podręcznej, to w przypadku dużych macierzy dostęp do elementów tej samej kolumny znajdujących się w innych wierszach prowadzi do nieefektywnego wykorzystania pamięci podręcznej. Warto w programie tak zorganizować dane w pamięci aby uniknąć opisanej sytuacji. Istnieje wiele opisanych w literaturze i Internecie schematów mnożenia macierzy, które w znacznym stopniu redukują nieefektywności związane z niezachowywaniem zasady lokalności dostępu.

Dodatkowo na obniżenie efektywności może wpłynąć rozrzutna gospodarka pamięcią dynamiczną. W szczególności należy unikać częstego alokowania i zwalniania niewielkich obszarów pamięci na stercie, gdyż zarządzanie pamięcią jest czasochłonne a ponadto może doprowadzić do niekorygowanej segmentacji pamięci w obszarze sterty, co z kolei może spowodować, że długotrwałe obliczenia doprowadzą do naruszenia ograniczenia na łączną pamięć alokowaną (10GB), pomimo, że rozmiar pamięci rzeczywiście wykorzystywanej będzie znacznie mniejszy.

Aby uniknąć w/w efektów w przypadku implementacji metod i funkcji operatorowych należy w szczególności zadbać o efektywną implementację konstruktorów kopiujących i operatorów przypisania.

UDANEJ ZABAWY!!!

Literatura

1. Fialko S. Modelowanie zagadnień technicznych. (Skrypt). Politechnika Krakowska, 2011 (https://torus.uck.pk.edu.pl/~fialko/text/MZT1/MZT.pdf)
2. Press W. H. et al., Numerical Recipes in C, Cambridge University Press, 2003 (dostępny jako pdf)
3. https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html, https://pages.nist.gov/hedgehog-Tutorials/tutorials/tutorial4.html, (dotyczy zabronionego GPU, ale opis problemu i metod również do wykorzystania w CPU)
4. https://gist.github.com/nadavrot/5b35d44e8ba3dd718e595e40184d03f0 (schemat efektywnego mnożenia macierzy)
5. https://en.wikipedia.org/wiki/Adjacency_matrix (zastosowanie operacji macierzowych dla dużych macierzy)
6. http://www.ams.org/publicoutreach/feature-column/fcarc-pagerank (zastosowanie operacji macierzowych dla dużych macierzy)
7. Meyers S. Język C++ bardziej efektywny, WNT, 1998 (pozycja trochę przestarzała, ale ogólne zasady pozostają aktualne).