Name: Jude Onyia
Student ID: V00947095
Email: judeonyia10@gmail.com
Course: ECE596C
Section: T01

Assignment ID: cpp_cache
Assignment Title: Cache-Efficient Algorithms

Submission Source: https://github.com/uvic-seng475-2020-05/cpp_cache-JudeOnyia.git
Commit ID: 91ba65b14354bca2c5f068d0d4abcb4ce2018dcd

Submitted Files
===============

```
drwxrwxr-x     107 2020-07-13 20:39 ./app
-rw-rw-r--     755 2020-07-13 20:39 ./app/test_fft.cpp
-rw-rw-r--    1826 2020-07-13 20:39 ./app/test_matrix_multiply.cpp
-rw-rw-r--    1974 2020-07-13 20:39 ./app/test_matrix_transpose.cpp
-rw-rw-r--     431 2020-07-13 20:39 ./CMakeLists.txt
-rw-rw-r--     139 2020-07-13 20:39 ./IDENTIFICATION.txt
drwxrwxr-x      24 2020-07-13 20:39 ./include
drwxrwxr-x      92 2020-07-13 20:39 ./include/ra
-rw-rw-r--    2260 2020-07-13 20:39 ./include/ra/fft.hpp
-rw-rw-r--    2734 2020-07-13 20:39 ./include/ra/matrix_multiply.hpp
-rw-rw-r--    2100 2020-07-13 20:39 ./include/ra/matrix_transpose.hpp
-rw-rw-r--  261158 2020-07-13 20:39 ./README.pdf
```

Results
=======

| Package | Operation | Target | Status |
|---|---|---|---|
| nonprog | generate | --- | OK (0.0s) |
| linalg_orig | generate | --- | OK (0.2s) |
| linalg_orig | configure | --- | FAIL (1 1.7s 18L) |
| linalg_orig | build | test_matrix_tran | ? (dependency) |
| linalg_orig | build | test_matrix_mult | ? (dependency) |
| linalg_sane | generate | --- | OK (0.4s) |
| linalg_sane | configure | --- | OK (1.8s) |
| linalg_sane | build | test_matrix_tran | OK (1.6s) |
| linalg_sane | build | test_matrix_mult | OK (1.0s) |
| fft_orig | generate | --- | OK (0.2s) |
| fft_orig | configure | --- | FAIL (1 1.6s 18L) |
| fft_orig | build | test_fft | ? (dependency) |
| fft_sane | generate | --- | OK (0.3s) |
| fft_sane | configure | --- | OK (3.4s) |
| fft_sane | build | test_fft | OK (3.8s) |

Normally, an operation is indicated as having a status of either "OK" or "FAIL". A status of "?" indicates that the operation could not be performed for some reason (e.g., due to an earlier error or being a manual step).
The time (in seconds) required for an operation is denoted by an expression consisting of a number followed by the letter "s" (e.g., "5.0s").
In the case of a test that consists of multiple test cases, the number of failed test cases and total number of test cases is expressed as a fraction (e.g., "10/50" means 10 test cases failed out of 50 test cases in total).
The length (in lines) of the log file generated by an operation is denoted by an expression consisting of a number followed by the letter "L" (e.g., "10L").

```
To ascertain the reason for the failure of an operation, check the contents
of the log file provided.

Legend
======

Package: nonprog
    Nonprogramming exercises

Package: linalg_orig
    The code as originally submitted by the student.
Build target: test_matrix_transpose
    Build the test_matrix_transpose program.
Build target: test_matrix_multiply
    Build the test_matrix_multiply program.

Package: linalg_sane
    Code with modifications to perform API sanity checking.
Build target: test_matrix_transpose
    Build the (dummy) test_matrix_transpose program.
Build target: test_matrix_multiply
    Build the (dummy) test_matrix_multiply program.

Package: fft_orig
    The code as originally submitted by the student.
Build target: test_fft
    Build the test_fft program.

Package: fft_sane
    Code with modifications to perform API sanity checking.
Build target: test_fft
    Build the (dummy) test_fft program.
```

```
CMake Error at CMakeLists.txt:1 (include):
  include could not find load file:

    Sanitizers.cmake


-- The CXX compiler identification is GNU 9.3.0
-- Check for working CXX compiler: /home/frodo/public/ugls_lab-4.0.70/bin/c++
-- Check for working CXX compiler: /home/frodo/public/ugls_lab-4.0.70/bin/c++ -
works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring incomplete, errors occurred!
See also
"/home/judeonyia/Documents/ECE596C_Assignments/ECE596C_Assgn_5/cpp_cache-JudeOny
ia/Assgn_5_precheck/package-linalg_orig/derived/CMakeFiles/CMakeOutput.log".
```

```
CMake Error at CMakeLists.txt:1 (include):
  include could not find load file:

    Sanitizers.cmake


-- The CXX compiler identification is GNU 9.3.0
-- Check for working CXX compiler: /home/frodo/public/ugls_lab-4.0.70/bin/c++
-- Check for working CXX compiler: /home/frodo/public/ugls_lab-4.0.70/bin/c++ -
works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring incomplete, errors occurred!
See also
"/home/judeonyia/Documents/ECE596C_Assignments/ECE596C_Assgn_5/cpp_cache-JudeOny
ia/Assgn_5_precheck/package-fft_orig/derived/CMakeFiles/CMakeOutput.log".
```

```
commit 15d69e013b139c25a40d5cee2889123a35cfb4e5
Author: JudeOnyia <60678029+JudeOnyia@users.noreply.github.com>
Date:   Sat Jul 11 20:59:10 2020 -0700

    First commit

commit 3cdc4960567c30c24aabd7395b5f375a76b3a43a
Author: Jude Onyia <judeonyia10@gmail.com>
Date:   Sun Jul 12 02:35:55 2020 -0700

    1) Wrote the base case and the recursive case
    2) Wrote the naive transpose function
    3) Tried using static variable to store original size of matrix

commit eeccb742236da012c83bb0bfe99353917ddf4ea4
Author: Jude Onyia <judeonyia10@gmail.com>
Date:   Sun Jul 12 15:43:58 2020 -0700

    1) Used static variables for original size of matrix
    2) Fix address overflow for case where matrix and result are different
    3) Made sure 2D matrix is all contiguous in memory

commit 508a80cea4b64e7db8dbd485de07b2d2a8f6f1fe
Author: Jude Onyia <judeonyia10@gmail.com>
Date:   Sun Jul 12 16:31:38 2020 -0700

    1) Fixed the condition for in-place transposition
    2) Wrote test for naive transposition
    3) Wrote test for in-place transposition

commit 1f30295b5ead7bd5e992dbe8969d3f3e5406c1e6
Author: Jude Onyia <judeonyia10@gmail.com>
Date:   Sun Jul 12 17:01:44 2020 -0700

    1) Removed static variables due case if function is used more than once
    2) Added a secondary function that takes a few more parameters needed

commit 7ef63e936daecec546f1df9ae86763b4bebf7882
Author: JudeOnyia <60678029+JudeOnyia@users.noreply.github.com>
Date:   Sun Jul 12 21:03:31 2020 -0700

    1) Wrote naive matrix multiply
    2) Wrote base case for matrix multiply
    3) Wrote the 3 cases, dividing M, or N, or P
    4) Used secondary recursive function approach
    5) Wrote test for both naive and efficient matrix multiplication

commit 2b169b0bf19517cd727ab9c72890f5a803e98bb0
Author: Jude Onyia <judeonyia10@gmail.com>
Date:   Sun Jul 12 23:45:47 2020 -0700

    1) Added lines to free memory in test cpp
    2) Performed some more testing

commit 2297423796b306dc9cf8d438be2699aaa64c41c7
Author: JudeOnyia <60678029+JudeOnyia@users.noreply.github.com>
Date:   Mon Jul 13 16:50:29 2020 -0700

    1) Wrote base case for fft function
    2) Wrote lines to factor n where n1 is close to sqrt(n)
    3) Transposed matrix x in-place using matrix_transpose class made earlier
    4) Replaced each row of x with n1 point dft of that row
```

```
63       5) Calculated and multiplied by twiddle factor
64       6) Transposed x in-place
65       7) Replaced each row of x with n2 point dft of that row
66       8) Transposed x in-place for correct order
67
68   commit 91ba65b14354bca2c5f068d0d4abcb4ce2018dcd
69   Author: Jude Onyia <judeonyia10@gmail.com>
70   Date:   Mon Jul 13 20:23:09 2020 -0700
71
72       1) Changed global constants pi,e,and j to static
73       2) changed format of input parameter to enable program to deduce
74          base class K
75       3) fixed typo in std::size_t spelling in main
76       4) fixed error declaring N as std::size_t
77       5) fixed error in using forward fft, namespace included
78       6) included <limits>
```

Name: Jude Onyia

Student ID: V00947095

Course: ECE 596C

Due Date: July 15, 2020

Assignment 5: Non – Programming Exercise

8.21)

$Block\ size = 64\ bytes = 2^6\ bytes$

$Therefore: \quad Block\ Offset = 6\ bits$

$Since\ 4-way\ set\ associative,\ each\ set = 4\ blocks = 4(64\ bytes) = 256\ bytes$

$Since\ capacity\ of\ cache = 32KB,\ the\ number\ of\ sets\ in\ cache = 32KB \times \frac{1\ set}{256\ bytes}$

$= 2^5 \times 2^{10} \times \frac{1\ set}{2^8} = 2^7\ sets = 128\ sets$

$Therefore: \quad Index = 7\ bits$

$Since\ the\ number\ of\ bits\ in\ an\ address\ is\ 32,\ Tag = 32 - 7 - 6 = 19\ bits$

$Tag = 19\ bits,\ Index = 7\ bits,\ Block\ Offset = 6\ bits$

8.20 a)

$Since\ index = 8\ bits\ \ and\ \ cache\ is\ 2-way\ associative,$

$Number\ of\ blocks\ in\ cache = Number\ of\ sets \times Number\ of\ blocks\ in\ a\ set$

$= 2^8 sets \times 2\ blocks\ per\ set = 512\ blocks$

8.20 b)

$Block\ addess = tag + index = 14\ bits + 8\ bits = 22\ bits$

$Therefore,\ number\ of\ blocks\ in\ memory = 2^{22} = 4194304\ blocks$

8.20 c)

$Address = 557A02_{16}$

$Tag = 0101\ 0101\ 0111\ 10_2$

$Index = 10\ 0000\ 00_2$

$Offset = 10_2$

This byte was present in the cache and its value is $C2_{16}$

8.20 d)

$Address = FFFFFF_{16}$

$Tag = 1111\ 1111\ 1111\ 11_2$

$Index = 11\ 1111\ 11_2$

$Offset = 11_2$

This byte was not present in the cache, no tags at index $11\ 1111\ 11_2$ matches its tag.

8.22)

Matrix $a$ of size 1024×1024 is aligned on a 64-byte boundary and the block size of the cache is 64 bytes, therefore, we do not need to worry about additional misses caused by alignment. An object of type double requires 8 bytes of storage, therefore each block of the cache can store: $64\ bytes \times \frac{1\ double}{8\ bytes} = 8\ elements\ of\ type\ double.$

Code fragment A accesses matrix $a$ in a cache efficient way, walking through the rows of matrix $a$. Since the language uses row major, the number of cache misses that occurs during the execution is: $\frac{Number\ of\ elements\ in\ matrix}{Number\ of\ elements\ a\ block\ can\ hold} = \frac{1024 \times 1024}{8} = 2^{17} = 131072\ cache\ misses.$

Code fragment B does not access the matrix in a cache efficient way. Fragment B walks through the columns of matrix $a$, requiring large strides in memory as it iterates. However, the capacity of the cache is 8KB. This results in $number\ of\ blocks = \frac{capacity}{block\ size} = \frac{8K\ bytes}{64\ bytes\ per\ block} = 128\ blocks$. Also, since the size of the column of the matrix is $1024\ elements \times \frac{1\ block}{8\ elements} = 128\ blocks$, the column of the matrix can indeed fit in the cache. Since the column of the matrix can fit in the cache, as the program goes down the first column (causing cache misses), the next column will result in cache hits because the blocks are still in the cache. Therefore, the number of cache misses that occurs during execution is: $\frac{Number\ of\ elements\ in\ matrix}{Number\ of\ elements\ a\ block\ can\ hold} = \frac{1024 \times 1024}{8} = 2^{17} = 131072\ cache\ misses.$

8.27)

The system has $page\ size = 1KB = 2^{10} bytes$, therefore, $Page\ offset = 10\ bits$. Since, virtual address is 24 bits, the virtual page number has $24 - 10 = 14\ bits$. The number of virtual pages is $2^{14} = 16384\ pages$. Since, the physical address is 16 bits, the physical page number has $16 - 10 = 6\ bits$. The number of physical pages is $2^6 = 64\ pages$.

8.28 a)

Virtual page number has 14 bits

Physical page number has 6 bits

Page offset has 10 bits

In virtual address $0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$, the virtual page number $0000\ 0000\ 0000\ 00_2$ is not in the page table. The protection check will result in an access violation.

8.28 b)

Virtual address $0000\ 0000\ 0000\ 1100\ 1100\ 1100_2$ maps to the physical address: $0010\ 1000\ 1100\ 1100_2$. The protection check will allow the write because this page is writable.

8.28 c)

Virtual address $1111\ 1111\ 1111\ 1100\ 0000\ 0000_2$ maps to the physical address: $0011\ 0000\ 0000\ 0000_2$. The protection check will result in an access violation because this page is not executable, so it cannot be fetched for execution.

```
include(Sanitizers.cmake)

# Specify Minimum Required Version
cmake_minimum_required(VERSION 3.1 FATAL_ERROR)

# Specify Project and Language
project(cpp_cache LANGUAGES CXX)

# Set Include Directory
include_directories(include)

# Add Executable Program
add_executable(test_matrix_transpose app/test_matrix_transpose.cpp)
add_executable(test_matrix_multiply app/test_matrix_multiply.cpp)
add_executable(test_fft app/test_fft.cpp)
```

```
#ifndef MATRIXTRANSPOSE
#define MATRIXTRANSPOSE
#include<cstddef>

namespace ra::cache {
    template <class T>
    void naive_matrix_transpose(const T* a, std::size_t m, std::size_t n, T* b){
        for(std::size_t i=0; i<m; ++i){
            for(std::size_t j=0; j<n; ++j){
                b[j*m+i] = a[i*n+j];
            }
        }
    }


    template <class T>
    void matrix_transpose_recurse(const T* a, std::size_t m, std::size_t n, T* b
, std::size_t M, std::size_t N){

        // If matrix a and b are the same, create a buffer matrix to store compu
tation
        bool flag_a_is_b = false;
        T* old_b = b; // old_b = b = a
        if(b == a){
            b = new T[m*n];
            flag_a_is_b = true;
        }

        // Base case
        if((m*n)<=64){
            for(std::size_t i=0; i<m; ++i){
                for(std::size_t j=0; j<n; ++j){
                    *(b+(j*M+i)) = *(a+(i*N+j));
                }
            }
        }
        // Recurse
        else{
            if(m >= n){ // Divide m
                std::size_t m1 = m / 2; // Number of rows in A1
                std::size_t m2 = m - m1; // Number of rows in A2
                const T* a1 = a; // pointer to first element in A1
                const T* a2 = a + (m1*N); // pointer to first element in A2
                T* b1 = b; // pointer to first element in B1
                T* b2 = b + m1; // pointer to first element in B2
                matrix_transpose_recurse(a1,m1,n,b1,M,N);
                matrix_transpose_recurse(a2,m2,n,b2,M,N);
            }
            else{ // Divide n
                std::size_t n1 = n / 2; // Number of columns in A1
                std::size_t n2 = n - n1; // Number of columns in A2
                const T* a1 = a; // pointer to the first element in A1
                const T* a2 = a + n1; // pointer to first element in A2
                T* b1 = b; // pointer to first element in B1
                T* b2 = b + (n1*M); // pointer to first element in B2
                matrix_transpose_recurse(a1,m,n1,b1,M,N);
                matrix_transpose_recurse(a2,m,n2,b2,M,N);
            }
        }

        // If matrix a and b were same and we created buffer matrix, copy back b
 to a
```

```
60          if(flag_a_is_b){
61              for(std::size_t i=0; i<(m*n); ++i){ // remember old_b = a
62                  *(old_b+i) = *(b+i);
63              }
64              delete[] b; // free buffer matrix
65          }
66      }
67
68
69      template <class T>
70      void matrix_transpose(const T* a, std::size_t m, std::size_t n, T* b){
71          matrix_transpose_recurse(a,m,n,b,m,n);
72      }
73
74
75
76  }
77  #endif
```

```cpp
#include "ra/matrix_transpose.hpp"
#include<iostream>
#include<cstddef>
#include<complex>

int main(){
    using std::cout;
    using std::endl;
    using type_t = double;

    std::size_t rows;
    std::size_t cols;
    rows = 10;
    cols = 17;

    // Creating matrix[rows][cols]
    type_t* matrx = new type_t[rows*cols];

    // Creating result[cols][rows]
    type_t* result = new type_t[cols*rows];

    // Creating naive_result[cols][rows]
    type_t* naive_result = new type_t[cols*rows];

    // Initialize matrix and result and naive result
    for(std::size_t i=0; i<rows; ++i){
        for(std::size_t j=0; j<cols; ++j){
            matrx[i*cols+j] = i*cols+j;
            result[j*rows+i] = 0;
            naive_result[j*rows+i] = 0;
        }
    }

    // TEST Transpose matrix where the original matrix and resulting matrix are
different
    cout << "TEST Transpose matrix (Different)"<<endl;
    ra::cache::matrix_transpose<type_t>(matrx,rows,cols,result);

    // Print matrix
    for(std::size_t i=0; i<rows; ++i){
        for(std::size_t j=0; j<cols; ++j){
            cout<<matrx[i*cols+j]<<" ";
        }
        cout<<endl;
    }
    cout<<endl;

    // Print result
    for(std::size_t i=0; i<cols; ++i){
        for(std::size_t j=0; j<rows; ++j){
            cout<<result[i*rows+j]<<" ";
        }
        cout<<endl;
    }
    cout<<endl;


    // TEST Naive_Transpose matrix
    cout << "TEST Naive Transpose matrix"<<endl;
    ra::cache::naive_matrix_transpose<type_t>(matrx,rows,cols,naive_result);

    // Print naive_result
```

```
62      for(std::size_t i=0; i<cols; ++i){
63          for(std::size_t j=0; j<rows; ++j){
64              cout<<naive_result[i*rows+j]<<" ";
65          }
66          cout<<endl;
67      }
68      cout<<endl;
69
70
71      // TEST Transpose matrix where the original matrix and resulting matrix are
    the same
72      cout<< "TEST Transpose matrix (in-place)" <<endl;
73      ra::cache::matrix_transpose<type_t>(matrx,rows,cols,matrx);
74
75      // Print matrix in-place transposed
76      for(std::size_t i=0; i<cols; ++i){
77          for(std::size_t j=0; j<rows; ++j){
78              cout<<matrx[i*rows+j]<<" ";
79          }
80          cout<<endl;
81      }
82
83      // Delete matrix and result
84      delete[] matrx;
85      delete[] result;
86      delete[] naive_result;
87
88
89
90
91  }
```

```
#ifndef MATRIXMULTIPLY
#define MATRIXMULTIPLY
#include<cstddef>

namespace ra::cache {
    template <class T>
    void naive_matrix_multiply(const T* a, const T* b, std::size_t m, std::size_t n, std::size_t p, T* c){
        for(std::size_t i=0; i<m; ++i){
            for(std::size_t j=0; j<p; ++j){
                T sum = T(0);
                for(std::size_t k=0; k<n; ++k){
                    sum += a[i*n+k] * b[k*p+j];
                }
                c[i*p+j] = sum;
            }
        }
    }

    template <class T>
    void matrix_multiply_recurse(const T* a, const T* b, std::size_t m, std::size_t n, std::size_t p, T* c, std::size_t M, std::size_t N, std::size_t P, bool ac_flag){
        // Base case
        if((m*n*p)<=64){
            for(std::size_t i=0; i<m; ++i){
                for(std::size_t j=0; j<p; ++j){
                    T sum = T(0);
                    for(std::size_t k=0; k<n; ++k){
                        sum += a[i*N+k] * b[k*P+j];
                    }
                    if(ac_flag){ c[i*P+j] += sum; }
                    else{ c[i*P+j] = sum; }
                }
            }
        }
        // Recurse
        else{
            if(m>=n && m>=p){ // Divide dimension m (Case 1)
                std::size_t m1 = m/2; // Number of rows in A1 and C1
                std::size_t m2 = m - m1; // Number of rows in A2 and C2
                const T* a1 = a; // pointer to first element in A1
                const T* a2 = a + (m1*N); // pointer to first element in A2
                T* c1 = c; // pointer to first element in C1
                T* c2 = c + (m1*P); // pointer to first element in C2
                matrix_multiply_recurse(a1, b, m1, n, p, c1, M, N, P, ac_flag);
                matrix_multiply_recurse(a2, b, m2, n, p, c2, M, N, P, ac_flag);
            }
            else if(n>=m && n>=p){ // Divide dimension n (Case 2)
                std::size_t n1 = n/2; // Number of columns in A1 and number of rows in B1
                std::size_t n2 = n - n1; // Number of columns in A2 and number of rows in B2
                const T* a1 = a; // pointer to first element in A1
                const T* a2 = a + n1; // pointer to first element in A2
                const T* b1 = b; // pointer to first element in B1
                const T* b2 = b + (n1*P); // pointer to first element in B2
                matrix_multiply_recurse(a1, b1, m, n1, p, c, M, N, P, ac_flag);
                matrix_multiply_recurse(a2, b2, m, n2, p, c, M, N, P, true);
            }
            else{ // Divide dimension p (Case 3)
                std::size_t p1 = p/2; // Number of columns in B1 and C1
```

```
58                  std::size_t p2 = p - p1; // Number of columns in B2 and C2
59                  const T* b1 = b; // pointer to first element in B1
60                  const T* b2 = b + p1; // pointer to first element in B2
61                  T* c1 = c; // pointer to first element in C1
62                  T* c2 = c + p1; // pointer to first element in C2
63                  matrix_multiply_recurse(a, b1, m, n, p1, c1, M, N, P, ac_flag);
64                  matrix_multiply_recurse(a, b2, m, n, p2, c2, M, N, P, ac_flag);
65              }
66          }
67      }
68
69
70      template <class T>
71      void matrix_multiply(const T* a, const T* b, std::size_t m, std::size_t n, s
    td::size_t p, T* c){
72          matrix_multiply_recurse(a,b,m,n,p,c,m,n,p,false);
73      }
74
75
76  }
77  #endif
```

```cpp
#include "ra/matrix_multiply.hpp"
#include<iostream>
#include<cstddef>
#include<complex>

int main(){
    using std::cout;
    using std::endl;
    using type_t = double;

    std::size_t M = 10;
    std::size_t N = 7;
    std::size_t P = 4;

    // Creating A[M][N]
    type_t* A = new type_t[M*N];

    // Creating B[N][P]
    type_t* B = new type_t[N*P];

    // Creating C[M][P]
    type_t* C = new type_t[M*P];

    // Creating naive_C[M][P]
    type_t* naive_C = new type_t[M*P];

    // Initialize A and display it
    cout<<"Matrix A"<<endl;
    for(std::size_t i=0; i<M; ++i){
        for(std::size_t j=0; j<N; ++j){
            A[i*N+j] = i*N+j;
            cout<<(A[i*N+j])<<" ";
        }
        cout<<endl;
    }
    cout<<endl;

    // Initialize B and display it
    cout<<"Matrix B"<<endl;
    for(std::size_t i=0; i<N; ++i){
        for(std::size_t j=0; j<P; ++j){
            B[i*P+j] = i*P+j;
            cout<<(B[i*P+j])<<" ";
        }
        cout<<endl;
    }
    cout<<endl;

    // Initialize C and naive_C and display C
    cout<<"Matrix C"<<endl;
    for(std::size_t i=0; i<M; ++i){
        for(std::size_t j=0; j<P; ++j){
            C[i*P+j] = i*P+j;
            cout<<(C[i*P+j])<<" ";
            naive_C[i*P+j] = i*P+j;
        }
        cout<<endl;
    }
    cout<<endl;

    // Test Matrix Multiplication
    cout << "Test Matrix Multiplication" << endl << endl;
```

```
63      ra::cache::matrix_multiply<type_t>(A, B, M, N, P, C);
64
65      // Test Naive Matrix Multiplication
66      cout << "Test Naive Matrix Multiplication" << endl << endl;
67      ra::cache::naive_matrix_multiply(A, B, M, N, P, naive_C);
68
69      // Display C
70      cout<<"Matrix C"<<endl;
71      for(std::size_t i=0; i<M; ++i){
72          for(std::size_t j=0; j<P; ++j){
73              cout<<(C[i*P+j])<<" ";
74          }
75          cout<<endl;
76      }
77      cout<<endl;
78
79      // Display naive_C
80      cout<<"Matrix naive_C"<<endl;
81      for(std::size_t i=0; i<M; ++i){
82          for(std::size_t j=0; j<P; ++j){
83              cout<<(naive_C[i*P+j])<<" ";
84          }
85          cout<<endl;
86      }
87      cout<<endl;
88
89      // Free
90      delete[] A;
91      delete[] B;
92      delete[] C;
93      delete[] naive_C;
94
95
96
97
98  }
```

```
#ifndef FFTHPP
#define FFTHPP
#include<cstddef>
#include "ra/matrix_transpose.hpp" // void matrix_transpose(const T* a, std::size_t m, std::size_t n, T* b)
#include<limits>
#include<boost/math/constants/constants.hpp>
#include<cmath> // std::sqrt(#)  and  std::log2(#)  and std::pow(base_size_t, power_size_t)
#include<complex> // std::pow(base_complex, power_complex)

namespace ra::cache {

    template <class K>
    void forward_fft(std::complex<K>* x, std::size_t n){
        using T = std::complex<K>;
        static K K_pi(boost::math::constants::pi<K>());
        static K K_e(boost::math::constants::e<K>());

        static T T_pi(K_pi,0);
        static T T_e(K_e,0);
        static T T_j = T(0,1);

        // base case
        if(n<=4){
            // if n = 0 or 1, do nothing b/c trivial. n cannot be 3 b/c n must be a power of 2
            if(n==2){
                T dft0 = x[0] + x[1];
                T dft1 = x[0] - x[1];
                x[0] = dft0; x[1] = dft1;
            }
            if(n==4){
                T dft0 = x[0] + x[1] + x[2] + x[3];
                T dft1 = x[0] - x[2] + (T_j * (x[3] - x[1]));
                T dft2 = x[0] - x[1] + x[2] - x[3];
                T dft3 = x[0] - x[2] + (T_j * (x[1] - x[3]));
                x[0] = dft0; x[1] = dft1; x[2] = dft2; x[3] = dft3;
            }
        }
        // Recurse case
        else{
            // factor n into n1*n2, where n1 is as close to sqrt(n) as possible
            std::size_t exp_of_n = std::log2(n);
            std::size_t exp_of_n1 = exp_of_n / 2;
            std::size_t exp_of_n2 = exp_of_n - exp_of_n1;
            std::size_t n1 = std::pow(2,exp_of_n1);
            std::size_t n2 = std::pow(2,exp_of_n2);

            matrix_transpose(x, n1, n2, x); // Treat x as a x[n1][n2] matrix and transpose it in-place to x[n2][n1]

            // Replace each row with n1-point DFT recursively
            for(std::size_t i=0; i<n2; ++i){
                T* xi = x + (i*n1);
                forward_fft(xi, n1);
            }

            // Muliply by twiddle factors
            T WN = std::pow(T_e, -(T_j * T(2,0) * T_pi / T(n,0)));
            for(std::size_t i=0; i<n2; ++i){
                for(std::size_t j=0; j<n1; ++j){
```

```
59                          x[i*n1+j] *= std::pow(WN, (i*j));
60                      }
61                  }
62
63                  matrix_transpose(x, n2, n1, x); // Transpose x[n2][n1] in-place to x
    [n1][n2]
64
65                  // Replace each row with n2-point DFT recursively
66                  for(std::size_t i=0; i<n1; ++i){
67                      T* xi = x + (i*n2);
68                      forward_fft(xi, n2);
69                  }
70
71                  matrix_transpose(x, n1, n2, x); // Transpose x to yield array with e
    lements in correct order
72              }
73          }
74
75
76      }
77      #endif
```

```cpp
#include "ra/fft.hpp"
#include<iostream>
#include<cstddef>
#include<complex>

int main(){
    /*
    using K = double;
    using T = std::complex<K>;
    K K_pi(boost::math::constants::pi<K>());
    K K_e(boost::math::constants::e<K>());
    T T_j = T(0,1);
    T T_pi = T(K_pi,0);
    T T_e = T(K_e,0);
    T WN = std::pow(T_e, -(T_j * T(2,0) * T_pi / T(4,0)));
    std::cout<<WN<<std::endl;
    */


    using std::cout;
    using std::endl;
    using type_t = std::complex<double>;

    std::size_t N = 32;
    type_t* x = new type_t[N]; // Create array x

    // Initialize array x
    for(std::size_t i=0; i<N; ++i){
        x[i] = i;
    }

    // Test fft function
    ra::cache::forward_fft(x,N);

    // display result
    for(std::size_t i=0; i<N; ++i){
        cout<<(x[i])<<" ";
    }
    cout<<endl;

    delete[] x; // Free x

}
```