```cpp
#include <iostream>
#include "mayer_primes.h"
#include <vector>
#include <algorithm>
#include "timing.h"
#include "omp.h"
using namespace std;



vector<unsigned int> count_goldbach(unsigned int n)
{
    vector<unsigned int> primes = get_primes(n);
    size_t npr = primes.size();
    int nthreads = omp_get_max_threads();

    vector<vector<unsigned int>> local_counts(nthreads, vector<unsigned int>(n+1, 0)
);

    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        auto &lc = local_counts[tid];

    #pragma omp for schedule(static)
        for (int j = 0; j < (int)npr; ++j)
        {
            unsigned int p = primes[j];
            for (unsigned int i = j;i < primes.size() && p + primes[i] <= n; ++i)
            {
                unsigned int q = primes[i];
                unsigned int s = p + q;

                lc[s] +=1;
            }
        }
    }

    // combine
    vector<unsigned int> counts(n+1, 0);
    for (int t = 0; t < nthreads; ++t)
        for (unsigned int k = 0; k <= n; ++k)
            counts[k] += local_counts[t][k];

    return counts;
}



int main()
{



    //3
    vector<unsigned int> counts = count_goldbach(100000);
    cout << (max_element(counts.begin(), counts.end()) - counts.begin()) << endl;

    //4
    vector<unsigned int> nvalues = {10000, 100000, 400000, 1000000, 2000000};
    double time;
    unsigned int n;
    for(unsigned int i = 0; i< nvalues.size(); i++)
    {
        n = nvalues.at(i);
        tic();
        count_goldbach(n);
```

```
   68:         time = toc();
   69:         cout << "Time for n=" << n << ": " << time << endl;
   70:     }
   71:
   72:
   73:
   74:
   75:
   76:     return 0;
   77: }
```

```
    1: #pragma once
    2:
    3: #include <cstring> //memset
    4: #include <vector>
    5: //using namespace std;
    6:
    7: /** \brief Determines all prime numbers in interval [2, @p max].
    8:  *
    9:  *  The sieve of Eratosthenes is used.
   10:  *
   11:  *  The implementation originates from <a href="http://code.activestate.com/recipes/
576559-fast-prime-generator/">Florian Mayer</a>.
   12:  *
   13:  * \param[in]   max end of interval for the prime number search.
   14:  * \return  vector of prime numbers @f$2,3,5, ..., p<=max @f$.
   15:  *
   16:  * \copyright
   17:  * Copyright (c) 2008 Florian Mayer          (adapted by Gundolf Haase 2018)
   18:  *
   19:  * Permission is hereby granted, free of charge, to any person obtaining a copy
   20:  * of this software and associated documentation files (the "Software"), to deal
   21:  * in the Software without restriction, including without limitation the rights
   22:  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
   23:  * copies of the Software, and to permit persons to whom the Software is
   24:  * furnished to do so, subject to the following conditions:
   25:  *
   26:  * The above copyright notice and this permission notice shall be included in
   27:  * all copies or substantial portions of the Software.
   28:  *
   29:  *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLI
ED,
   30:  * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
   31:  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
   32:  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
   33:  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
   34:  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOF
TWARE.
   35:  *
   36:  */
   37: template <class T>
   38: std::vector<T> get_primes(T max)
   39: {
   40:     std::vector<T> primes;
   41:     char *sieve;
   42:     sieve = new char[max / 8 + 1];
   43:     // Fill sieve with 1
   44:     memset(sieve, 0xFF, (max / 8 + 1) * sizeof(char));
   45:     for (T x = 2; x <= max; x++)
   46:     {
   47:         if (sieve[x / 8] & (0x01 << (x % 8))) {
   48:             primes.push_back(x);
   49:             // Is prime. Mark multiplicates.
   50:             for (T j = 2 * x; j <= max; j += x)
   51:             {
   52:                 sieve[j / 8] &= ~(0x01 << (j % 8));
   53:                         }
   54:         }
   55:         }
   56:     delete[] sieve;
   57:     return primes;
   58: }
   59:
   60: //----------------------------------------------------------------
   61: //int main()        // by  Florian Mayer
   62: //{g++ -O3  -std=c++14 -fopenmp main.cpp && ./a.out
   63: //    vector<unsigned long> primes;
   64: //    primes = get_primes(10000000);
   65: //    // return 0;
```

```
66: //    // Print out result.
67: //    vector<unsigned long>::iterator it;
68: //    for(it=primes.begin(); it < primes.end(); it++)
69: //        cout << *it << " ";
70: //
71: //    cout << endl;
72: //    return 0;
73: //}
74:
```

```
    1: //
    2: //    Gundolf Haase, Oct 18 2024
    3: //
    4: #pragma once
    5: #include <chrono>                   // timing
    6: #include <stack>
    7:
    8: //using Clock = std::chrono::system_clock;   //!< The wall clock timer chosen
    9: using Clock = std::chrono::high_resolution_clock;
   10: using TPoint= std::chrono::time_point<Clock>;
   11:
   12: // [Galowicz, C++17 STL Cookbook, p. 29]
   13:
   14: std::stack<TPoint> MyStopWatch; //!< starting time of stopwatch
   15:
   16: /** Starts stopwatch timer.
   17:  *  Use as @code tic(); myfunction(...) ; double tsec = toc();  @endcode
   18:  *
   19:  *  The timining can be nested and the recent time point is stored on top of the sta
ck.
   20:  *
   21:  *  @return recent time point
   22:  *  @see toc
   23:  */
   24:  auto tic()
   25: {
   26:     MyStopWatch.push(Clock::now());
   27:     return MyStopWatch.top();
   28: }
   29:
   30: /** Returns the elapsed time from stopwatch.
   31:  *
   32:  * The time point from top of the stack is used
   33:  * if time point @p t_b is not passed as input parameter.
   34:  * Use as @code tic(); myfunction(...) ; double tsec = toc();  @endcode
   35:  * or as @code auto t_b = tic(); myfunction(...) ; double tsec = toc(t_b);  @endcode
   36:  * The last option is to be used in the case of
   37:  * non-nested but overlapping time measurements.
   38:  *
   39:  * @param[in]  t_b start time of some stop watch
   40:  * @return elapsed time in seconds.
   41:  *
   42: */
   43:  double toc(TPoint const &t_b = MyStopWatch.top())
   44: {
   45:     // https://en.cppreference.com/w/cpp/chrono/treat_as_floating_point
   46:     using Unit       = std::chrono::seconds;
   47:     using FpSeconds = std::chrono::duration<double, Unit::period>;
   48:     auto t_e = Clock::now();
   49:     MyStopWatch.pop();
   50:     return FpSeconds(t_e-t_b).count();
   51: }
```