```cpp
#include "goldbach.h"
#include <iostream>
#include <iterator>
#include <omp.h>

size_t single_goldbach(size_t k)
{
    const std::vector<size_t> relevant_primes = get_primes(k);
    size_t m = relevant_primes.size();

    size_t counter = 0;

#pragma omp parallel for shared(relevant_primes, m, k) reduction(+:counter)
    for(size_t i = 0; i < m; ++i)
    {
        for(size_t j = i; j < m; ++j)
        {
            if(relevant_primes[i] + relevant_primes[j] == k)
                ++counter;
        }
    }

    return counter;
}


std::vector<size_t> count_goldbach(size_t n)
{
    const std::vector<size_t> relevant_primes = get_primes(n);
    size_t m = relevant_primes.size();

    std::vector<size_t> counter_vector(n + 1, 0);

#pragma omp parallel for shared(relevant_primes, m, n) reduction(VecAdd:counter_vector)
    for(size_t i = 0; i < m; ++i)
    {
        for(size_t j = i; j < m; ++j)
        {
            size_t sum = relevant_primes[i] + relevant_primes[j];
            if(sum <= n)
                ++counter_vector[relevant_primes[i] + relevant_primes[j]];
        }
    }

    return counter_vector;
}
```

```
    1: #pragma once
    2: #include "mayer_primes.h"
    3: #include <cassert>
    4: #include <vector>
    5:
    6:
    7: /**
    8:   This function returns the number of possible decompositions of an integer into a s
um of two prime numbers.
    9:   @param[in]    k            first integer
   10:   @param[out]   count        number of decompositions
   11: */
   12: size_t single_goldbach(size_t k);
   13:
   14:
   15: /**
   16:   This function returns the number of possible decompositions into a sum of two prim
e numbers of all even integers in the interval [4,n].
   17:   @param[in]    n                   upper integer bound
   18:   @param[out]   count_vector        vector containing the number of decompositions f
or a natural number the corresponding index
   19: */
   20: std::vector<size_t> count_goldbach(size_t n);
   21:
   22:
   23: /**      Vector @p b adds its elements to vector @p a .
   24:         @param[in] a    vector
   25:         @param[in] b    vector
   26:         @return             a+=b componentwise
   27: */
   28: template<class T>
   29: std::vector<T> &operator+=(std::vector<T> &a, std::vector<T> const &b)
   30: {
   31:     assert(a.size()==b.size());
   32:     for (size_t k = 0; k < a.size(); ++k) {
   33:         a[k] += b[k];
   34:     }
   35:     return a;
   36: }
   37:
   38: // Declare the reduction operation in OpenMP for an STL-vector
   39: //   omp_out += omp_in  requires operator+=(vector<int> &, vector<int> const &) from
 above
   40: // ------------------------------------------------------------
   41: // https://scc.ustc.edu.cn/zlsc/tc4600/intel/2016.0.109/compiler_c/common/core/GUID-
7312910C-D175-4544-99C5-29C12D980744.htm
   42: // https://gist.github.com/eruffaldi/7180bdec4c8c9a11f019dd0ba9a2d68c
   43: // https://stackoverflow.com/questions/29633531/user-defined-reduction-on-vector-of-
varying-size
   44: //  see also p.74ff in  https://www.fz-juelich.de/ias/jsc/EN/AboutUs/Staff/Hagemeier
_A/docs-parallel-programming/OpenMP-Slides.pdf
   45: #pragma omp declare reduction(VecAdd : std::vector<size_t>  : omp_out += omp_in) ini
tializer (omp_priv=omp_orig)
```

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


int main()
{
    cout << "Check: 694 has "<< single_goldbach(694) << " decompositions." << endl << "----------------------------------------" << endl;

    for(size_t n : {10000, 100000, 400000, 1000000, 2000000})
    {
        double t_start = omp_get_wtime();

        auto goldbach_vector = count_goldbach(n);

        auto max_it = max_element(goldbach_vector.begin(), goldbach_vector.end());
        size_t max_number = distance(goldbach_vector.begin(), max_it);

        double t_end = omp_get_wtime() - t_start;

        cout << "The number " <<  max_number << " has " << *max_it << " decompositions. Duration: " << t_end << endl;
    }

    /*
    ###### WITHOUT PARALLELIZATION ######
    The number 9240 has 329 decompositions. Duration: 0.00307696
    The number 99330 has 2168 decompositions. Duration: 0.189839
    The number 390390 has 7094 decompositions. Duration: 1.3042
    The number 990990 has 15594 decompositions. Duration: 5.45034
    The number 1981980 has 27988 decompositions. Duration: 47.1807


    ###### WITH PARALLELIZATION ######
    The number 9240 has 329 decompositions. Duration: 0.000734854
    The number 99330 has 2168 decompositions. Duration: 0.0251322
    The number 390390 has 7094 decompositions. Duration: 0.487375
    The number 990990 has 15594 decompositions. Duration: 6.16972
    The number 1981980 has 27988 decompositions. Duration: 31.5699
    */


    return 0;
}
```

```
    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:
```