```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;


// Inner product
double benchmark_A(const vector<double> &x, const vector<double> &y)
{
    double sum = 0.0;
    #pragma omp parallel for reduction(+:sum)
     for (unsigned int i = 0; i < x.size(); i++)
    {
        sum += x[i]*y[i];
    }
    return sum;
}

// Inner product
double benchmark_A_sum(const vector<double> &x)
{
    double sum = 0.0;
    #pragma omp parallel for reduction(+:sum)
     for (unsigned int i = 0; i < x.size(); i++)
    {
        sum += x[i];
    }
    return sum;
}

//Matrix-vector product
vector<double> benchmark_B(const vector<double> &A, const vector<double> &x)
{
    unsigned int N = x.size();
    unsigned int M = A.size() / N;
    vector<double> b(M, 0.0);

    #pragma omp parallel for
    for (unsigned int i = 0; i < M; i++)
    {
        double bi = 0.0;
        for (unsigned int j = 0; j < N; j++)
        {
            bi += A[i*N+j]*x[j];
        }
        b[i] = bi;
    }

    return b;
}


//Matrix-Matrix product
vector<double> benchmark_C(const vector<double> &A, const vector<double> &B, unsigned int M)
{
    unsigned int L = A.size()/M;
    unsigned int N = B.size()/L;
    vector<double> C(M*N,0.0);
    #pragma omp parallel for collapse(2)
     for (unsigned int i = 0; i < M; i++)
    {
        for (unsigned int j = 0; j < N; j++)
        {
            double sum = 0.0;
            for (unsigned int k = 0; k < L; k++)
            {
                sum += A[i*L+k]*B[k*N+j];
```

```
68:             }
69:             C[i*N+j] = sum;
70:         }
71:     }
72:
73:     return C;
74:
75: }
76:
77: //polynomial evaluation
78: vector<double> benchmark_D(const vector<double>& coeff, const vector<double>& x)
79: {
80:     unsigned int p  = coeff.size();   // p coefficients, degree p-1
81:     unsigned int N  = x.size();
82:     vector<double> y(N);
83:
84: #pragma omp parallel for
85:     for (unsigned int i = 0; i < N; i++){
86:         double yi = coeff[p-1];
87:         double xi = x[i];
88:         for(int j=p-2; j>=0; --j)
89:         {
90:             yi = yi*xi+coeff[j];
91:         }
92:         y[i] = yi;
93:     }
94:     return y;
95: }
96:
97:
98: double benchmark_A_old(const vector<double> &x, const vector<double> &y)
99: {
100:     double sum = 0.0;
101:      for (unsigned int i = 0; i < x.size(); i++)
102:     {
103:         sum += x[i]*y[i];
104:     }
105:     return sum;
106: }
107: double benchmark_A_sum_old(const vector<double> &x)
108: {
109:     double sum = 0.0;
110:      for (unsigned int i = 0; i < x.size(); i++)
111:     {
112:         sum += x[i];
113:     }
114:     return sum;
115: }
```

```
 1: #ifndef BENCHMARK_H
 2: #define BENCHMARK_H
 3:
 4: #include <vector>
 5: using namespace std;
 6:
 7: double benchmark_A(const vector<double> &x,
 8:                    const vector<double> &y);
 9:     double benchmark_A_sum(const vector<double> &x);
10:
11: vector<double> benchmark_B(const vector<double> &A,
12:                               const vector<double> &x);
13:
14: vector<double> benchmark_C(const vector<double> &A,
15:                               const vector<double> &B,
16:                               unsigned int M);
17:
18: vector<double> benchmark_D(const vector<double> &coefficients,
19:                               const vector<double> &x);
20: double benchmark_A_old(const vector<double> &x,
21:                    const vector<double> &y);
22:     double benchmark_A_sum_old(const vector<double> &x);
23:
24:
25:
26:
27: #endif
```

```
    1: #include "mylib.h"
    2: #include <cassert>
    3: #include <chrono>            // timing
    4: #include <cmath>             // sqrt()
    5: #include <cstdlib>           // atoi()
    6: #include <cstring>           // strncmp()
    7: #include <ctime>
    8: #include <iostream>
    9: #include <sstream>
   10: #include "benchmark.h"
   11: #include "omp.h"
   12: using namespace std;
   13: using namespace std::chrono;  // timing
   14:
   15: int main(int argc, char **argv)
   16: {
   17:     const unsigned int NA = 1400000;
   18:     const unsigned int NLOOPSA = 2000;
   19:     //const unsigned int NLOOPS = 10;
   20:
   21:     const unsigned int MC = 1000;
   22:     int const NLOOPSC = 5;
   23: // ---------- Benchmark A ----------
   24:
   25: {
   26:
   27:
   28:     vector<double> xA(NA), yA(NA);
   29:     for (unsigned int i = 0; i < NA; ++i)
   30:     {
   31:         double xi= (i % 219) + 1;
   32:         xA[i] = xi;
   33:         yA[i] = 1.0 / xi;
   34:     }
   35:
   36:     auto tA1 = system_clock::now();
   37:     double sA = 0.0, sumA = 0.0;
   38:     for (unsigned int loop = 0; loop < NLOOPSA; ++loop)
   39:     {
   40:         sA = benchmark_A(xA, yA);
   41:         sumA += sA;
   42:     }
   43:     auto tA2 = system_clock::now();
   44:
   45:     auto durA = duration_cast<microseconds>(tA2 - tA1);
   46:     double tA = static_cast<double>(durA.count()) / 1e6 / NLOOPSA; //duration per lo
op seconds
   47:
   48:     cout << "\n===== Benchmark A =====\n";
   49:     cout << "<xA,yA> = "  << sA << endl;
   50:     cout << "Timing in sec. : " << tA << endl;
   51:     cout << "GFLOPS         : " << 2.0 * NA / tA / 1024 / 1024 / 1024 << endl;
   52:     cout << "GiByte/s       : "
   53:          << 2.0 * NA * sizeof(xA[0]) / tA / 1024 / 1024 / 1024 << endl;
   54: }
   55:
   56: // ---------- Benchmark B----------
   57:
   58: {
   59:     const unsigned int MB = 1700;
   60:     const unsigned int NB = MB;
   61:     const unsigned int NLOOPSB = 200;//50;
   62:
   63:     vector<double> AB(MB * NB);
   64:     vector<double> xB(NB);
   65:
   66:     for (unsigned int i = 0; i < MB; ++i)
   67:         for (unsigned int j = 0; j < NB; ++j)
```

```
    68:             AB[i * NB + j] = (i+j) %219 +1;
    69:
    70:     for (unsigned int j = 0; j < NB; ++j)
    71:     {
    72:
    73:         xB[j] = 1.0 / AB[17*NB+j];
    74:     }
    75:
    76:     vector<double> bB;
    77:     auto tB1 = system_clock::now();
    78:     double guardB = 0.0;
    79:     for (unsigned int loop = 0; loop < NLOOPSB; ++loop)
    80:     {
    81:         bB = benchmark_B(AB, xB);
    82:         guardB += bB[17];
    83:     }
    84:     auto tB2 = system_clock::now();
    85:
    86:     auto durB = duration_cast<microseconds>(tB2 - tB1);
    87:     double tB = static_cast<double>(durB.count()) / 1e6 / NLOOPSB;
    88:
    89:     double flopsB = 2.0 * MB * NB;
    90:     double bytesB = (MB * NB + NB  + MB)  * sizeof(double);
    91:
    92:     cout << "\n===== Benchmark B =====\n";
    93:     cout << guardB << endl;
    94:     cout << "bytes: " << bytesB << endl;
    95:     cout << "Timing in sec. : " << tB << endl;
    96:     cout << "GFLOPS         : " << flopsB / tB / 1024 / 1024 / 1024 << endl;
    97:     cout << "GiByte/s       : " << bytesB / tB / 1024 / 1024 / 1024 << endl;
    98: }
    99:
   100: // ---------- Benchmark C ----------
   101:
   102: {
   103:
   104:     const unsigned int LC = MC;
   105:     const unsigned int NC = MC;
   106:
   107:
   108:     vector<double> AC(MC * LC), BC(LC * NC);
   109:
   110:     for (unsigned int i = 0; i < MC; ++i)
   111:         for (unsigned int j = 0; j < LC; ++j)
   112:             AC[i * LC + j] = (i+j) %219 +1;
   113:
   114:     for (unsigned int i = 0; i < LC; ++i)
   115:         for (unsigned int j = 0; j < NC; ++j)
   116:             BC[i * NC + j] = (i+j) %219 +1;
   117:
   118:     vector<double> CC;
   119:     auto tC1 = system_clock::now();
   120:     double guardC = 0.0;
   121:     for (unsigned int loop = 0; loop < NLOOPSC; ++loop)
   122:     {
   123:         CC = benchmark_C(AC, BC, MC);
   124:         guardC += CC[0];
   125:     }
   126:     auto tC2 = system_clock::now();
   127:
   128:     auto durC = duration_cast<microseconds>(tC2 - tC1);
   129:     double tC = static_cast<double>(durC.count()) / 1e6 / NLOOPSC;
   130:
   131:     double flopsC = 2.0 * MC * LC * NC;
   132:     double bytesC = (MC * LC  + LC * NC  + MC * NC)* sizeof(double);
   133:
   134:     cout << "\n===== Benchmark C =====\n";
   135:     cout << guardC << endl;
```

```
136:     cout << "bytes: " << bytesC << endl;
137:     cout << "Timing in sec. : " << tC << endl;
138:     cout << "GFLOPS         : " << flopsC / tC / 1024 / 1024 / 1024 << endl;
139:     cout << "GiByte/s       : " << bytesC / tC / 1024 / 1024 / 1024 << endl;
140: }
141:
142: // ---------- Benchmark D----------
143:
144:
145: {
146:     const unsigned int ND = 2000000;
147:     const unsigned int p  = 14;          // degree p-1 = 15
148:     const unsigned int NLOOPSD = 100;
149:     vector<double> coeff(p, 0.0);
150:     vector<double> xD(ND);
151:
152:     for (unsigned int k = 0; k < p; ++k)
153:         coeff[k] = k%219+1;
154:
155:     for (unsigned int i = 0; i < ND; ++i)
156:         xD[i] = i%219+1;
157:
158:     vector<double> yD;
159:     auto tD1 = system_clock::now();
160:     double guardD = 0.0;
161:     for (unsigned int loop = 0; loop < NLOOPSD; ++loop)
162:     {
163:
164:         yD = benchmark_D(coeff, xD);
165:         guardD += yD[0];
166:     }
167:     auto tD2 = system_clock::now();
168:
169:     auto durD = duration_cast<microseconds>(tD2 - tD1);
170:     double tD = static_cast<double>(durD.count()) / 1e6 / NLOOPSD;
171:
172:
173:     double flopsD = ND * 2 * p;
174:     double bytesD = (p + 2 * ND)*sizeof(double);
175:
176:     cout << "\n===== Benchmark D =====\n";
177:     cout << guardD << endl;
178:     cout << "bytes: " << bytesD << endl;
179:     cout << "Timing in sec. : " << tD << endl;
180:     cout << "GFLOPS         : " << flopsD / tD / 1024 / 1024 / 1024 << endl;
181:     cout << "GiByte/s       : " << bytesD / tD / 1024 / 1024 / 1024 << endl;
182: }
183:
184:
185:
186: const int NLOOPS = 20;
187:
188: for (int k = 3; k <= 8; ++k)
189: {
190:     unsigned int n = pow(10.0, k);
191:
192:     vector<double> x(n), y(n);
193:     for (unsigned int  i = 0; i < n; ++i)
194:     {
195:         double xi = (i % 219) + 1;
196:         x[i] = xi;
197:         y[i] = 1.0 / xi;
198:     }
199:
200:     double s1_guard = 0.0, s2_guard = 0.0;
201:     double ip1_guard = 0.0, ip2_guard = 0.0;
202:
203:     // ---- SUM benchmark (sequential) ----
```

```
204:     double t0 = omp_get_wtime();
205:     for (int r = 0; r < NLOOPS; ++r)
206:         s1_guard += benchmark_A_sum_old(x);
207:     double t_sum_seq = (omp_get_wtime() - t0) / NLOOPS;
208:
209:     // ---- SUM benchmark (parallel) ----
210:     t0 = omp_get_wtime();
211:     for (int r = 0; r < NLOOPS; ++r)
212:         s2_guard += benchmark_A_sum(x);
213:     double t_sum_omp = (omp_get_wtime() - t0) / NLOOPS;
214:
215:     double sum_speedup = t_sum_seq / t_sum_omp;
216:
217:     // ---- INNER PRODUCT benchmark (sequential) ----
218:     t0 = omp_get_wtime();
219:     for (int r = 0; r < NLOOPS; ++r)
220:         ip1_guard += benchmark_A_old(x, y);
221:     double t_inner_seq = (omp_get_wtime() - t0) / NLOOPS;
222:
223:     // ---- INNER PRODUCT benchmark (parallel) ----
224:     t0 = omp_get_wtime();
225:     for (int r = 0; r < NLOOPS; ++r)
226:         ip2_guard += benchmark_A(x, y);
227:     double t_inner_omp = (omp_get_wtime() - t0) / NLOOPS;
228:
229:     double inner_speedup = t_inner_seq / t_inner_omp;
230:
231:     // ---- Print results ----
232:     std::cout << "k = " << k << " (n = 10^" << k << " = " << n << ")\n";
233:     std::cout << "SUM    seq: " << t_sum_seq   << " s,  omp: " << t_sum_omp
234:               << " s,  speedup = " << sum_speedup   << '\n';
235:     std::cout << "INNER  seq: " << t_inner_seq << " s,  omp: " << t_inner_omp
236:               << " s,  speedup = " << inner_speedup << '\n';
237:     std::cout << "guards: "
238:               << s1_guard << ", " << s2_guard << ", "
239:               << ip1_guard << ", " << ip2_guard << "\n\n";
240: }
241:
242:
243:
244:
245: return 0;
246: }
247:
```

```cpp
#include "mylib.h"
#include <cassert>       // assert()
#include <cmath>
#include <vector>

#ifdef __INTEL_CLANG_COMPILER
#pragma message(" ##########  Use of MKL  ###############")
#include <mkl.h>
#else
#pragma message(" ##########  Use of CBLAS  ###############")
//extern "C"
//{
#include <cblas.h>                // cBLAS Library
#include <lapacke.h>              // Lapack
//}
#endif

using namespace std;

double scalar(vector<double> const &x, vector<double> const &y)
{
    assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
    size_t const N = x.size();
    double sum = 0.0;
    for (size_t i = 0; i < N; ++i)
    {
        sum += x[i] * y[i];
        //sum += exp(x[i])*log(y[i]);
    }
    return sum;
}


double scalar_cblas(vector<double> const &x, vector<double> const &y)
{
    int const asize = static_cast<int>(size(x));
    int const bsize = static_cast<int>(size(y));
    assert(asize == bsize); // switch off via compile flag: -DNDEBUG
        return cblas_ddot(asize,x.data(),1,y.data(),1);
    //assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
        //return cblas_ddot(x.size(),x.data(),1,y.data(),1);
}

float scalar_cblas(vector<float> const &x, vector<float> const &y)
{
    int const asize = static_cast<int>(size(x));
    int const bsize = static_cast<int>(size(y));
    assert(asize == bsize); // switch off via compile flag: -DNDEBUG
        return cblas_sdot(asize,x.data(),1,y.data(),1);
    //assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
        //return cblas_ddot(x.size(),x.data(),1,y.data(),1);
}


double norm(vector<double> const &x)
{
    size_t const N = x.size();
    double sum = 0.0;
    for (size_t i = 0; i < N; ++i)
    {
        sum += x[i] * x[i];
    }
    return std::sqrt(sum);
}

```

```
 1: #ifndef FILE_MYLIB
 2: #define FILE_MYLIB
 3: #include <vector>
 4:
 5: /**     Inner product
 6:         @param[in] x    vector
 7:         @param[in] y    vector
 8:         @return             resulting Euclidian inner product <x,y>
 9: */
10: double scalar(std::vector<double> const &x, std::vector<double> const &y);
11:
12: /**     Inner product using BLAS routines
13:         @param[in] x    vector
14:         @param[in] y    vector
15:         @return             resulting Euclidian inner product <x,y>
16: */
17: double scalar_cblas(std::vector<double> const &x, std::vector<double> const &y);
18: float scalar_cblas(std::vector<float> const &x, std::vector<float> const &y);
19:
20:
21: /**     L_2 Norm of a vector
22:         @param[in] x    vector
23:         @return             resulting Euclidian norm <x,y>
24: */
25: double norm(std::vector<double> const &x);
26:
27:
28:
29:
30: #endif
```