```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;
     for (unsigned int i = 0; i < x.size(); i++)
    {
        sum += x[i]*y[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);

    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);
     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];
            }
            C[i*N+j] = sum;
        }
    }

    return C;

}

//polynomial evaluation
vector<double> benchmark_D(const vector<double>& coeff, const vector<double>& x)
{
    unsigned int p  = coeff.size();    // p coefficients, degree p-1
    unsigned int N  = x.size();
    vector<double> y(N);
```

swap j- and k loop
That should be faster.

```
   68:
   69:     for (unsigned int i = 0; i < N; i++){
   70:         double yi = coeff[p-1];
   71:         double xi = x[i];
   72:         for(int j=p-2; j>=0; --j)
   73:         {
   74:             yi = yi*xi+coeff[j];
   75:         }
   76:         y[i] = yi;
   77:     }
   78:     return y;
   79: }
   80:
   81: //TASK 5
   82: double norm2(const vector<double>& x)
   83: {
   84:     double s = 0.0;
   85:     double xi;
   86:     for (unsigned int i = 0; i < x.size(); ++i){
   87:         xi = x[i];
   88:         s += xi*xi;
   89:     }
   90:
   91:     return sqrt(s);
   92: }
   93:
   94: double scalar_kahan(const vector<double>& x, const vector<double>& y)
   95: {
   96:     double sum = 0.0;
   97:     double c   = 0.0;
   98:
   99:     for (unsigned int i = 0; i < x.size(); i++)
  100:     {
  101:         double prod = x[i]*y[i];
  102:         double yk = prod - c;
  103:         double t  = sum+yk;
  104:         c= (t - sum) - yk;
  105:         sum = t;
  106:     }
  107:     return sum;
  108: }
  109:
  110: //Matrix-Matrix product
  111: vector<double> matrixMultColumnWise(const vector<double> &A, const vector<double> &B
, unsigned int M)
  112: {
  113:     unsigned int L = A.size()/M;
  114:     unsigned int N = B.size()/L;
  115:     vector<double> C(M*N,0.0);
  116:      for (unsigned int i = 0; i < M; i++)
  117:     {
  118:         for (unsigned int j = 0; j < N; j++)
  119:         {
  120:             double sum = 0.0;
  121:             for (unsigned int k = 0; k < L; k++)
  122:             {
  123:                 sum += A[k*M+i]*B[k*N+j];
  124:             }
  125:             C[i*N+j] = sum;
  126:         }
  127:     }
  128:
  129:     return C;
  130:
  131: }
```

Best loop order

k

i

j

```
 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:
10: vector<double> benchmark_B(const vector<double> &A,
11:                               const vector<double> &x);
12:
13: vector<double> benchmark_C(const vector<double> &A,
14:                               const vector<double> &B,
15:                               unsigned int M);
16:
17: vector<double> benchmark_D(const vector<double> &coefficients,
18:                               const vector<double> &x);
19:
20:
21: double norm2(const vector<double>& x);
22: double scalar_kahan(const vector<double>& x, const vector<double>& y);
23:
24: vector<double> matrixMultColumnWise(const vector<double> &A,
25:                               const vector<double> &B,
26:                               unsigned int M);
27:
28:
29: #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: using namespace std;
   12: using namespace std::chrono;  // timing
   13:
   14: int main(int argc, char **argv)
   15: {
   16:     const unsigned int NA = 1400000;
   17:     const unsigned int NLOOPSA = 2000;
   18:     //const unsigned int NLOOPS = 10;
   19:
   20:     const unsigned int MC = 1000;
   21:     int const NLOOPSC = 5;
   22: // ---------- Benchmark A ----------
   23:
   24: {
   25:
   26:
   27:     vector<double> xA(NA), yA(NA);
   28:     for (unsigned int i = 0; i < NA; ++i)
   29:     {
   30:         double xi= (i % 219) + 1;
   31:         xA[i] = xi;
   32:         yA[i] = 1.0 / xi;
   33:     }
   34:
   35:     auto tA1 = system_clock::now();
   36:     double sA = 0.0, sumA = 0.0;
   37:     for (unsigned int loop = 0; loop < NLOOPSA; ++loop)
   38:     {
   39:         sA = benchmark_A(xA, yA);
   40:         sumA += sA;
   41:     }
   42:     auto tA2 = system_clock::now();
   43:
   44:     auto durA = duration_cast<microseconds>(tA2 - tA1);
   45:     double tA = static_cast<double>(durA.count()) / 1e6 / NLOOPSA; //duration per lo
op seconds
   46:
   47:     cout << "\n===== Benchmark A =====\n";
   48:     cout << "<xA,yA> = "  << sA << endl;
   49:     cout << "Timing in sec. : " << tA << endl;
   50:     cout << "GFLOPS         : " << 2.0 * NA / tA / 1024 / 1024 / 1024 << endl;
   51:     cout << "GiByte/s       : "
   52:          << 2.0 * NA * sizeof(xA[0]) / tA / 1024 / 1024 / 1024 << endl;
   53: }
   54:
   55: // ---------- Benchmark B----------
   56:
   57: {
   58:     const unsigned int MB = 1700;
   59:     const unsigned int NB = MB;
   60:     const unsigned int NLOOPSB = 200;//50;
   61:
   62:     vector<double> AB(MB * NB);
   63:     vector<double> xB(NB);
   64:
   65:     for (unsigned int i = 0; i < MB; ++i)
   66:         for (unsigned int j = 0; j < NB; ++j)
   67:             AB[i * NB + j] = (i+j) %219 +1;
```

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

```
136:     cout << "Timing in sec. : " << tC << endl;
137:     cout << "GFLOPS         : " << flopsC / tC / 1024 / 1024 / 1024 << endl;
138:     cout << "GiByte/s       : " << bytesC / tC / 1024 / 1024 / 1024 << endl;
139: }
140:
141: // ---------- Benchmark D----------
142:
143:
144: {
145:     const unsigned int ND = 2000000;
146:     const unsigned int p  = 14;          // degree p-1 = 15
147:     const unsigned int NLOOPSD = 100;
148:     vector<double> coeff(p, 0.0);
149:     vector<double> xD(ND);
150:
151:     for (unsigned int k = 0; k < p; ++k)
152:         coeff[k] = k%219+1;
153:
154:     for (unsigned int i = 0; i < ND; ++i)
155:         xD[i] = i%219+1;
156:
157:     vector<double> yD;
158:     auto tD1 = system_clock::now();
159:     double guardD = 0.0;
160:     for (unsigned int loop = 0; loop < NLOOPSD; ++loop)
161:     {
162:
163:         yD = benchmark_D(coeff, xD);
164:         guardD += yD[0];
165:     }
166:     auto tD2 = system_clock::now();
167:
168:     auto durD = duration_cast<microseconds>(tD2 - tD1);
169:     double tD = static_cast<double>(durD.count()) / 1e6 / NLOOPSD;
170:
171:
172:     double flopsD = ND * 2 * p;
173:     double bytesD = (p + 2 * ND)*sizeof(double);
174:
175:     cout << "\n===== Benchmark D =====\n";
176:     cout << guardD << endl;
177:     cout << "bytes: " << bytesD << endl;
178:     cout << "Timing in sec. : " << tD << endl;
179:     cout << "GFLOPS         : " << flopsD / tD / 1024 / 1024 / 1024 << endl;
180:     cout << "GiByte/s       : " << bytesD / tD / 1024 / 1024 / 1024 << endl;
181: }
182:
183:
184: //-------------TASK 5
185: {
186:
187:
188:     vector<double> xA(NA);
189:     for (unsigned int i = 0; i < NA; ++i)
190:     {
191:         double xi= (i % 219) + 1;
192:         xA[i] = xi;
193:     }
194:
195:     auto tA1 = system_clock::now();
196:     double sA = 0.0, sumA = 0.0;
197:     for (unsigned int loop = 0; loop < NLOOPSA; ++loop)
198:     {
199:         sA = norm2(xA);
200:         sumA += sA;
201:     }
202:     auto tA2 = system_clock::now();
203:
```

```
  204:     auto durA = duration_cast<microseconds>(tA2 - tA1);
  205:     double tA = static_cast<double>(durA.count()) / 1e6 / NLOOPSA; //duration per lo
op seconds
  206:
  207:     cout << "\n===== Benchmark 5A =====\n";
  208:     cout << "NORM = " << sA << endl;
  209:     cout << "Timing in sec. : " << tA << endl;
  210:     cout << "GFLOPS         : " << 2.0 * NA / tA / 1024 / 1024 / 1024 << endl;
  211:     cout << "GiByte/s       : "
  212:          <<  NA * sizeof(xA[0]) / tA / 1024 / 1024 / 1024 << endl;
  213:
  214:     //a bit faster due to only accessing one vector
  215: }
  216:
  217:
  218: {
  219:
  220:
  221:     vector<double> xA(NA), yA(NA);
  222:     for (unsigned int i = 0; i < NA; ++i)
  223:     {
  224:         double xi= (i % 219) + 1;
  225:         xA[i] = xi;
  226:         yA[i] = 1.0 / xi;
  227:     }
  228:
  229:     auto tA1 = system_clock::now();
  230:     double sA = 0.0, sumA = 0.0;
  231:     for (unsigned int loop = 0; loop < NLOOPSA; ++loop)
  232:     {
  233:         sA = scalar_kahan(xA, yA);
  234:         sumA += sA;
  235:     }
  236:     auto tA2 = system_clock::now();
  237:
  238:     auto durA = duration_cast<microseconds>(tA2 - tA1);
  239:     double tA = static_cast<double>(durA.count()) / 1e6 / NLOOPSA; //duration per lo
op seconds
  240:
  241:     cout << "\n===== Benchmark 5B =====\n";
  242:     cout << "<xA,yA> = " << sA << endl;
  243:     cout << "Timing in sec. : " << tA << endl;
  244:     cout << "GFLOPS         : " << 5.0 * NA / tA / 1024 / 1024 / 1024 << endl;
  245:     cout << "GiByte/s       : "
  246:          << 2.0 * NA * sizeof(xA[0]) / tA / 1024 / 1024 / 1024 << endl;
  247:
  248:     //in comparison to benchmark A: a bit slower runtime but more than double the am
ount of FLOPS therefor also more GFLOPS
  249: }
  250:
  251:
  252:
  253: {
  254:
  255:     const unsigned int LC = MC;
  256:     const unsigned int NC = MC;
  257:
  258:     vector<double> AC(MC * LC), BC(LC * NC);
  259:
  260:     for (unsigned int i = 0; i < MC; ++i)
  261:         for (unsigned int j = 0; j < LC; ++j)
  262:             AC[i * LC + j] = (i+j) %219 +1;
  263:
  264:     for (unsigned int i = 0; i < LC; ++i)
  265:         for (unsigned int j = 0; j < NC; ++j)
  266:             BC[i * NC + j] = (i+j) %219 +1;
  267:
  268:     vector<double> CC;
```

```
  269:     auto tC1 = system_clock::now();
  270:     double guardC = 0.0;
  271:     for (unsigned int loop = 0; loop < NLOOPSC; ++loop)
  272:     {
  273:         CC = matrixMultColumnWise(AC, BC, MC);
  274:         guardC += CC[0];
  275:     }
  276:     auto tC2 = system_clock::now();
  277:
  278:     auto durC = duration_cast<microseconds>(tC2 - tC1);
  279:     double tC = static_cast<double>(durC.count()) / 1e6 / NLOOPSC;
  280:
  281:     double flopsC = 2.0 * MC * LC * NC;
  282:     double bytesC = (MC * LC  + LC * NC  + MC * NC)* sizeof(double);
  283:
  284:     cout << "\n===== Benchmark 5C =====\n";
  285:     cout << guardC << endl;
  286:     cout << "bytes: " << bytesC << endl;
  287:     cout << "Timing in sec. : " << tC << endl;
  288:     cout << "GFLOPS         : " << flopsC / tC / 1024 / 1024 / 1024 << endl;
  289:     cout << "GiByte/s       : " << bytesC / tC / 1024 / 1024 / 1024 << endl;
  290:
  291:     //slower than rowwise access, due to incoherent acces in the vector memory of A
  292:
  293:     //Transpose matrix, the it is also row wise-access or reorder loops
  294: }
  295:
  296: return 0;
  297: }
```

```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
```

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

// Inner product
double benchmark_A(const vector<double> &x, const vector<double> &y)
{


    return cblas_ddot(x.size(),x.data(),1,y.data(),1);

}

//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);

    cblas_dgemv(CblasRowMajor,CblasNoTrans,M,N,1,A.data(),N,x.data(),1,0.0,b.data(),1);

    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);

    cblas_dgemm(CblasRowMajor,CblasNoTrans,CblasNoTrans,M,N,L,1.0,A.data(),L,B.data(),N,0.0,C.data(),N);

    return C;

}
```

```
 1: #ifndef BENCHMARK_H
 2: #define BENCHMARK_H
 3:
 4:
 5: #include <vector>
 6: using namespace std;
 7:
 8: double benchmark_A(const vector<double> &x,
 9:                    const vector<double> &y);
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:
19:
20:
21: #endif
```

```cpp
#include "mylib.h"
#include <cassert>
#include <chrono>            // timing
#include <cmath>             // sqrt()
#include <cstdlib>           // atoi()
#include <cstring>           // strncmp()
#include <ctime>
#include <iostream>
#include <sstream>
#include "benchmark.h"
using namespace std;
using namespace std::chrono;  // timing

int main(int argc, char **argv)
{
    const unsigned int NA = 1400000;
    const unsigned int NLOOPSA = 10000;
    //const unsigned int NLOOPS = 10;

    const unsigned int MC = 1000;
    int const NLOOPSC = 1000;
// ---------- Benchmark A ----------

{


    vector<double> xA(NA), yA(NA);
    for (unsigned int i = 0; i < NA; ++i)
    {
        double xi= (i % 219) + 1;
        xA[i] = xi;
        yA[i] = 1.0 / xi;
    }

    auto tA1 = system_clock::now();
    double sA = 0.0, sumA = 0.0;
    for (unsigned int loop = 0; loop < NLOOPSA; ++loop)
    {
        sA = benchmark_A(xA, yA);
        sumA += sA;
    }
    auto tA2 = system_clock::now();

    auto durA = duration_cast<microseconds>(tA2 - tA1);
    double tA = static_cast<double>(durA.count()) / 1e6 / NLOOPSA; //duration per lo
op seconds

    cout << "\n===== Benchmark A =====\n";
    cout << "<xA,yA> = "  << sA << endl;
    cout << "Timing in sec. : " << tA << endl;
    cout << "GFLOPS         : " << 2.0 * NA / tA / 1024 / 1024 / 1024 << endl;
    cout << "GiByte/s       : "
         << 2.0 * NA * sizeof(xA[0]) / tA / 1024 / 1024 / 1024 << endl;
}

// ---------- Benchmark B----------

{
    const unsigned int MB = 1700;
    const unsigned int NB = MB;
    const unsigned int NLOOPSB = 10000;//50;

    vector<double> AB(MB * NB);
    vector<double> xB(NB);

    for (unsigned int i = 0; i < MB; ++i)
        for (unsigned int j = 0; j < NB; ++j)
            AB[i * NB + j] = (i+j) %219 +1;
```

```
    for (unsigned int j = 0; j < NB; ++j)
    {

        xB[j] = 1.0 / AB[17*NB+j];
    }

    vector<double> bB;
    auto tB1 = system_clock::now();
    double guardB = 0.0;
    for (unsigned int loop = 0; loop < NLOOPSB; ++loop)
    {
        bB = benchmark_B(AB, xB);
        guardB += bB[17];
    }
    auto tB2 = system_clock::now();

    auto durB = duration_cast<microseconds>(tB2 - tB1);
    double tB = static_cast<double>(durB.count()) / 1e6 / NLOOPSB;

    double flopsB = 2.0 * MB * NB;
    double bytesB = (MB * NB + NB  + MB)  * sizeof(double);

    cout << "\n===== Benchmark B =====\n";
    cout << guardB << endl;
    cout << "bytes: " << bytesB << endl;
    cout << "Timing in sec. : " << tB << endl;
    cout << "GFLOPS         : " << flopsB / tB / 1024 / 1024 / 1024 << endl;
    cout << "GiByte/s       : " << bytesB / tB / 1024 / 1024 / 1024 << endl;
}

// ---------- Benchmark C ----------

{

    const unsigned int LC = MC;
    const unsigned int NC = MC;


    vector<double> AC(MC * LC), BC(LC * NC);

    for (unsigned int i = 0; i < MC; ++i)
        for (unsigned int j = 0; j < LC; ++j)
            AC[i * LC + j] = (i+j) %219 +1;

    for (unsigned int i = 0; i < LC; ++i)
        for (unsigned int j = 0; j < NC; ++j)
            BC[i * NC + j] = (i+j) %219 +1;

    vector<double> CC;
    auto tC1 = system_clock::now();
    double guardC = 0.0;
    for (unsigned int loop = 0; loop < NLOOPSC; ++loop)
    {
        CC = benchmark_C(AC, BC, MC);
        guardC += CC[0];
    }
    auto tC2 = system_clock::now();

    auto durC = duration_cast<microseconds>(tC2 - tC1);
    double tC = static_cast<double>(durC.count()) / 1e6 / NLOOPSC;

    double flopsC = 2.0 * MC * LC * NC;
    double bytesC = (MC * LC  + LC * NC  + MC * NC)* sizeof(double);

    cout << "\n===== Benchmark C =====\n";
    cout << "bytes: " << bytesC << endl;
    cout << "Timing in sec. : " << tC << endl;
```

```
136:     cout << "GFLOPS          : " << flopsC / tC / 1024 / 1024 / 1024 << endl;
137:     cout << "GiByte/s        : " << bytesC / tC / 1024 / 1024 / 1024 << endl;
138: }
139:
140:
141: return 0;
142: }
```

```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
```

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

// Inner product
double benchmark_A(const vector<double> &x, const vector<double> &y)
{


    return cblas_ddot(x.size(),x.data(),1,y.data(),1);

}

//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);

    cblas_dgemv(CblasRowMajor,CblasNoTrans,M,N,1,A.data(),N,x.data(),1,0.0,b.data(),1);

    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);

    cblas_dgemm(CblasRowMajor,CblasNoTrans,CblasNoTrans,M,N,L,1.0,A.data(),L,B.data(),N,0.0,C.data(),N);

    return C;

}


```

```
 1: #ifndef BENCHMARK_H
 2: #define BENCHMARK_H
 3:
 4:
 5: #include <vector>
 6: using namespace std;
 7:
 8: double benchmark_A(const vector<double> &x,
 9:                    const vector<double> &y);
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:
19:
20:
21: #endif
```

```cpp
#include <cassert>
#include <chrono>            // timing
#include <cmath>             // sqrt()
#include <cstdlib>           // atoi()
#include <cstring>           // strncmp()
#include <ctime>
#include <iostream>
#include <sstream>
#include <vector>
#include <lapacke.h>
#include "timing.h"
#include "benchmark.h"
using namespace std;
using namespace std::chrono;  // timing

int main()
{
    unsigned int n= 32;

    vector<double> M(n*n,4.0);

    for(unsigned int i=0; i<n; i++)
    {
        for(unsigned int j=0; j<n; j++)
        {
            if(i!=j)
            {
                double diff = i-j;
                M[i*n+j] = 1.0/(diff*diff);
            }
        }


    }
    vector<double> M2 = M;



    vector<int> ipiv(n); //pivots
    LAPACKE_dgetrf(LAPACK_ROW_MAJOR,n,n, M.data(),n,ipiv.data());  //M=PLU



    double time;
    unsigned int nhrsmax = 1000000;
    for(unsigned int i=nhrsmax/10; i < nhrsmax;i+=nhrsmax/10)
    {

        unsigned int nhrs = i;

        //FOR CHECKING
        vector<double> X(n*nhrs,1.0);

        vector<double> b = benchmark_C(M2,X,n);

        tic();
        LAPACKE_dgetrs(LAPACK_ROW_MAJOR,'N',n,nhrs,M.data(),n,ipiv.data(),b.data(),n
hrs);
        time = toc();
        cout << "Time for nhrs=" << nhrs << ": " << time << endl;



        double max_err = 0.0;
        for (unsigned int j = 0; j < n * nhrs; j++)
        {
            double err = b[j] - X[j];
```

timing: large matrices are needed.

```
   68:             err *= err;
   69:             if (err > max_err) max_err = err;
   70:         }
   71:         cout <<"max err^2:" << max_err <<endl;
   72:         cout <<endl;
   73:
   74:     }
   75:         /*
   76:
   77:         Time for nhrs=100000: 0.0605495
   78:         max err^2:4.93038e-32
   79:
   80:         Time for nhrs=200000: 0.127608
   81:         max err^2:4.93038e-32
   82:
   83:         Time for nhrs=300000: 0.182197
   84:         max err^2:4.93038e-32
   85:
   86:         Time for nhrs=400000: 0.202608
   87:         max err^2:4.93038e-32
   88:
   89:         Time for nhrs=500000: 0.24484
   90:         max err^2:4.93038e-32
   91:
   92:         Time for nhrs=600000: 0.298055
   93:         max err^2:4.93038e-32
   94:
   95:         Time for nhrs=700000: 0.362414
   96:         max err^2:4.93038e-32
   97:
   98:         Time for nhrs=800000: 0.410004
   99:         max err^2:4.93038e-32
  100:
  101:         Time for nhrs=900000: 0.492339
  102:         max err^2:4.93038e-32
  103:
  104:         Time grows slow (linearly)
  105:     */
  106:
  107:
  108:
  109:
  110:
  111:
  112:     return 0;
  113:
  114:
  115: }
```

```
    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: }
```