```
    1: #include "benchmarks.h"
    2: #include <cassert>       // assert()
    3: #include <cmath>
    4: #include <iostream>
    5: #include <vector>
    6: #include <omp.h>
    7:
    8: // (A) Inner product of two vectors (from skalar_stl)
    9: double scalar_parallel(vector<double> const &x, vector<double> const &y)
   10: {
   11:     assert(x.size() == y.size());
   12:     size_t const N = x.size();
   13:     double sum = 0.0;
   14: //#pragma omp parallel for default(none) shared(x, y, N) reduction(+:sum) schedule(r
untime)
   15: #pragma omp parallel for shared(x, y, N) reduction(+:sum)
   16:     for (size_t i = 0; i < N; ++i)
   17:     {
   18:         sum += x[i] * y[i];
   19:     }
   20:     return sum;
   21: }
   22:
   23: // (A) Vector entry sum
   24: double sum(vector<double> const &x)
   25: {
   26:     double sum = 0.0;
   27: #pragma omp parallel for shared(x) reduction(+:sum)
   28:     for (size_t i = 0; i < x.size(); ++i)
   29:     {
   30:         sum += x[i];
   31:     }
   32:     return sum;
   33: }
   34:
   35:
   36: // (B) Matrix-vector product (from intro_vector_densematrix)
   37: vector<double> MatVec_parallel(vector<double> const &A, vector<double> const &x)
   38: {
   39:     size_t const nelem = A.size();
   40:     size_t const N = x.size();
   41:     assert(nelem % N == 0); // make sure multiplication is possible
   42:     size_t const M = nelem/N;
   43:
   44:     vector<double> b(M);
   45:
   46: #pragma omp parallel for shared(A, x, N, M, b)
   47:     for (size_t i = 0; i < M; ++i)
   48:     {
   49:         double tmp = 0.0;
   50:         for (size_t j = 0; j < N; ++j)
   51:             tmp += A[N*i + j] * x[j];
   52:         b[i] = tmp;
   53:     }
   54:
   55:     return b;
   56: }
   57:
   58:
   59: // (C) Matrix-matrix product
   60: vector<double> MatMat_parallel(vector<double> const &A, vector<double> const &B, siz
e_t const &L)
   61: {
   62:     size_t const nelem_A = A.size();
   63:     size_t const nelem_B = B.size();
   64:
   65:     assert(nelem_A % L == 0 && nelem_B % L == 0);
   66:
```

```
    size_t const M = nelem_A/L;
    size_t const N = nelem_B/L;


    vector<double> C(M*N);


#pragma omp parallel for shared(A, B, M, N, L, C)
    for (size_t i = 0; i < M; ++i)
    {
        for (size_t k = 0; k < L; ++k)
        {
            for (size_t j = 0; j < N; ++j)
            {
                C[N*i + j] += A[L*i + k]*B[N*k + j];
            }

        }
    }

    return C;
}


// (D) Evaluation of a polynomial function
vector<double> poly_parallel(vector<double> const &a, vector<double> const &x)
{
    size_t const N = x.size();
    size_t const p = a.size() - 1;
    vector<double> y(N, 0);

#pragma omp parallel for shared(a, x, N, p, y)
    for (size_t i = 0; i < N; ++i)
    {
        double x_temp = x[i];
        double y_temp = 0;
        for (size_t k = 0; k < p + 1; ++k)
        {
            y_temp += x_temp*y_temp + a[p - k];
        }
        y[i] = y_temp;
    }

    return y;
}
```

```
  135:
  136:
  137:
  138:
  139:
  140:
  141:
```

```
    1: #pragma once
    2: #include <vector>
    3: using namespace std;
    4:
    5: /**      (A) Inner product of two vectors (from skalar_stl)
    6:          @param[in] x    vector
    7:          @param[in] y    vector
    8:          @return             resulting Euclidian inner product <x,y>
    9: */
   10: double scalar_parallel(vector<double> const &x, vector<double> const &y);
   11:
   12:
   13: /**      (A) Sum entries of vector
   14:          @param[in] x    vector
   15:          @return             sum
   16: */
   17: double sum(vector<double> const &x);
   18:
   19:
   20: /**      (B) Matrix-vector product (from intro_vector_densematrix)
   21:  *       @param[in] A    dense matrix (1D access)
   22:  *  @param[in] u         vector
   23:  *
   24:  *       @return    resulting vector
   25: */
   26: vector<double> MatVec_parallel(vector<double> const &A, vector<double> const &x);
   27:
   28:
   29: /**      (C) Matrix-matrix product
   30:  *       @param[in] A                    MxL dense matrix (1D access)
   31:  *  @param[in] B                         LxN dense matrix (1D access)
   32:  *  @param[in] shared_dim        shared dimension L
   33:  *
   34:  *       @return    resulting MxN matrix
   35: */
   36: vector<double> MatMat_parallel(vector<double> const &A, vector<double> const &B, siz
e_t const &shared_dim);
   37:
   38:
   39: /**      (D) Evaluation of a polynomial function using Horner's scheme
   40:  *       @param[in] a    coefficient vector
   41:  *  @param[in] x         vector with input values
   42:  *
   43:  *       @return    vector with output values
   44: */
   45: vector<double> poly_parallel(vector<double> const &a, vector<double> const &x);
   46:
   47:
   48:
   49:
   50:
   51:
   52:
   53:
   54:
   55:
```

```
    1: #include "benchmark_tests.h"
    2: #include "benchmarks.h"
    3: #include <chrono>
    4: #include <iostream>
    5: #include <math.h>
    6: using namespace std::chrono;
    7:
    8: vector<double> test_A(const size_t &NLOOPS, const size_t &N)
    9: {
   10:     cout << "##################### (A) ####################" << endl;
   11:     cout << "\nLOOPS = " << NLOOPS << endl;
   12:     cout << "\nN = " << N << endl;
   13:
   14:
   15: //  Memory allocation
   16:     cout << "Memory allocation\n";
   17:
   18:     vector<double> x(N), y(N);
   19:
   20:     cout.precision(2);
   21:     cout << 2.0*N *sizeof(x[0]) / 1024 / 1024 / 1024 << " GByte Memory allocated\n";
   22:     cout.precision(6);
   23:
   24:
   25: //  Data initialization
   26: //  Special:  x_i = i+1;  y_i = 1/x_i  ==> <x,y> == N
   27:
   28:     for (size_t i = 0; i < N; ++i)
   29:     {
   30:         x[i] = i % 219 + 1;
   31:         y[i] = 1.0/x[i];
   32:     }
   33:
   34:
   35:     cout << "\nStart Benchmarking scalar\n";
   36:
   37:     auto t1 = system_clock::now(); // start timer
   38: // Do calculation
   39:     double check(0.0),ss(0.0);
   40:     for (size_t i = 0; i < NLOOPS; ++i)
   41:     {
   42:         check = scalar_parallel(x, y);
   43:         ss += check;                    // prevents the optimizer from removing unuse
d calculation results.
   44:     }
   45:
   46:     auto t2 = system_clock::now();  // stop timer
   47:     auto duration = duration_cast<microseconds>(t2 - t1);        // duration in micr
oseconds
   48:     double t_diff = static_cast<double>(duration.count()) / 1e6; // overall duration
 in seconds
   49:     t_diff = t_diff/NLOOPS;                                      // duration per loo
p seconds
   50:
   51:
   52:
   53: // Check the correct result
   54:     cout << "\n <x,y> = " << check << endl;
   55:     if (static_cast<unsigned int>(check) != N)
   56:         cout << "  !!   W R O N G  result   !!\n";
   57:     cout << endl;
   58:
   59:
   60: // Timings  and Performance
   61:     cout << endl;
   62:     cout.precision(2);
   63:
   64:
```

```
    double Gflops = 2.0*N / t_diff / 1024 / 1024 / 1024;
    double MemBandwidth = 2.0*N / t_diff / 1024 / 1024 / 1024 * sizeof(x[0]);

    cout << "Total duration : " << t_diff*NLOOPS << endl;
    cout << "Timing in sec. : " << t_diff << endl;
    cout << "GFLOPS         : " << Gflops << endl;
    cout << "GiByte/s       : " << MemBandwidth << endl;


    return vector<double>{t_diff, Gflops, MemBandwidth};
}

vector<double> test_A_sum(const size_t &NLOOPS, const size_t &N)
{
    cout << "#################### (A) sum ####################" << endl;
    cout << "\nLOOPS = " << NLOOPS << endl;
    cout << "\nN = " << N << endl;


//  Memory allocation
    cout << "Memory allocation\n";

    vector<double> x(N);

    cout.precision(2);
    cout << 1.0*N *sizeof(x[0]) / 1024 / 1024 / 1024 << " GByte Memory allocated\n";
    cout.precision(6);


//  Data initialization

    for (size_t i = 0; i < N; ++i)
    {
        x[i] = 1;
    }


    cout << "\nStart Benchmarking sum\n";

    auto t1 = system_clock::now(); // start timer
// Do calculation
    double check(0.0),ss(0.0);
    for (size_t i = 0; i < NLOOPS; ++i)
    {
        check = sum(x);
        ss += check;                    // prevents the optimizer from removing unused calculation results.
    }

    auto t2 = system_clock::now();  // stop timer
    auto duration = duration_cast<microseconds>(t2 - t1);        // duration in microseconds
    double t_diff = static_cast<double>(duration.count()) / 1e6; // overall duration in seconds
    t_diff = t_diff/NLOOPS;                                      // duration per loop seconds


// Check the correct result
    cout << "\n <x,y> = " << check << endl;
    if (static_cast<unsigned int>(check) != N)
        cout << "  !!   W R O N G  result   !!\n";
    cout << endl;


// Timings  and Performance
    cout << endl;
```

```
  129:     cout.precision(2);
  130:
  131:
  132:     double Gflops = 1.0*N / t_diff / 1024 / 1024 / 1024;
  133:     double MemBandwidth = 1.0*N / t_diff / 1024 / 1024 / 1024 * sizeof(x[0]);
  134:
  135:     cout << "Total duration : " << t_diff*NLOOPS << endl;
  136:     cout << "Timing in sec. : " << t_diff << endl;
  137:     cout << "GFLOPS         : " << Gflops << endl;
  138:     cout << "GiByte/s       : " << MemBandwidth << endl;
  139:
  140:
  141:     return vector<double>{t_diff, Gflops, MemBandwidth};
  142: }
  143:
  144:
  145: vector<double> test_B(const size_t &NLOOPS, const size_t &N, const size_t &M)
  146: {
  147:     cout << "#################### (B) ####################" << endl;
  148:
  149:     cout << "\nLOOPS = " << NLOOPS << endl;
  150:     cout << "\nN = " << N << endl;
  151:     cout << "\nM = " << M << endl;
  152:
  153: //  Memory allocation
  154:     cout << "Memory allocation\n";
  155:
  156:     vector<double> A(M*N);
  157:     vector<double> x(N);
  158:
  159:     cout.precision(2);
  160:     cout << (1.0*M*N + N) * sizeof(x[0]) / 1024 / 1024 / 1024 << " GByte Memory allo
cated\n";
  161:     cout.precision(6);
  162:
  163: //  Data initialization
  164:
  165:     for (size_t i = 0; i < M; ++i)
  166:         for (size_t j = 0; j < N; ++j)
  167:             A[N*i + j] = (i + j) % 219 + 1;
  168:
  169:
  170:     for (size_t j = 0; j < N; ++j)
  171:     {
  172:         x[j] = 1.0/A[N*17 + j];
  173:     }
  174:
  175:     cout << "\nStart Benchmarking MatVec\n";
  176:
  177:     auto t1 = system_clock::now(); // start timer
  178: // Do calculation
  179:     vector<double> b(M);
  180:
  181:     for (size_t i = 0; i < NLOOPS; ++i)
  182:     {
  183:         b = MatVec_parallel(A, x);
  184:     }
  185:
  186:     auto t2 = system_clock::now();  // stop timer
  187:     auto duration = duration_cast<microseconds>(t2 - t1);         // duration in micr
oseconds
  188:     double t_diff = static_cast<double>(duration.count()) / 1e6; // overall duration
 in seconds
  189:     t_diff = t_diff/NLOOPS;                                      // duration per loo
p seconds
  190:
  191:
  192: // Check the correct result
```

```
  193:     cout << "\n <A[17,*],x> = " << b[17] << endl;
  194:     if (static_cast<size_t>(b[17]) != N)
  195:     {
  196:         cout << "  !!   W R O N G  result   !!\n";
  197:     }
  198:     cout << endl;
  199:
  200:
  201: // Timings  and Performance
  202:     cout << endl;
  203:     cout.precision(2);
  204:
  205:     double Gflops = (2.0*N*M) / t_diff / 1024 / 1024 / 1024;
  206:     double MemBandwidth = (2.0*N*M + M)/ t_diff / 1024 / 1024 / 1024 * sizeof(x[0]);
  207:
  208:     cout << "Total duration : " << t_diff*NLOOPS << endl;
  209:     cout << "Timing in sec. : " << t_diff << endl;
  210:     cout << "GFLOPS         : " << Gflops << endl;
  211:     cout << "GiByte/s       : " << MemBandwidth << endl;
  212:
  213:
  214:
  215:     return vector<double>{t_diff, Gflops, MemBandwidth};
  216: }
  217:
  218:
  219: vector<double> test_C(const size_t &NLOOPS, const size_t &L, const size_t &M, const
size_t &N)
  220: {
  221:     cout << "#################### (C) ####################" << endl;
  222:     cout << "\nLOOPS = " << NLOOPS << endl;
  223:     cout << "\nL = " << L << endl;
  224:     cout << "\nM = " << M << endl;
  225:     cout << "\nN = " << N << endl;
  226:
  227:
  228: //  Memory allocation
  229:     cout << "Memory allocation\n";
  230:
  231:     vector<double> A(M*L);
  232:     vector<double> B(L*N);
  233:
  234:     cout.precision(2);
  235:     cout << (1.0*M*L + L*N) *sizeof(A[0]) / 1024 / 1024 / 1024 << " GByte Memory all
ocated\n";
  236:     cout.precision(6);
  237:
  238:
  239: //  Data initialization
  240:
  241:     for (size_t i = 0; i < M; ++i)
  242:         for (size_t k = 0; k < L; ++k)
  243:             A[L*i + k] = (i + k) % 219 + 1;
  244:
  245:     for (size_t k = 0; k < L; ++k)
  246:         for (size_t j = 0; j < N; ++j)
  247:             B[N*k + j] = 1.0/A[L*17 + k];
  248:
  249:
  250:     cout << "\nStart Benchmarking MatMat\n";
  251:
  252:     auto t1 = system_clock::now(); // start timer
  253: // Do calculation
  254:     vector<double> C(M*N);
  255:     double check;
  256:     double check_sum = 0;
  257:
  258:     for (size_t i = 0; i < NLOOPS; ++i)
```

```
  259:     {
  260:         C = MatMat_parallel(A, B, L);
  261:
  262:         check = C[N*17];
  263:         check_sum += check; // prevents the optimizer from removing unused calculati
on results.
  264:     }
  265:     cout << check_sum;
  266:
  267:     auto t2 = system_clock::now();  // stop timer
  268:     auto duration = duration_cast<microseconds>(t2 - t1);        // duration in micr
oseconds
  269:     double t_diff = static_cast<double>(duration.count()) / 1e6; // overall duration
 in seconds
  270:     t_diff = t_diff/NLOOPS;                                      // duration per loo
p seconds
  271:
  272:
  273: // Check the correct result
  274:     cout << "\n C[17,0] = " << check << endl;
  275:     if (static_cast<unsigned int>(check) != L)
  276:     {
  277:         cout << "  !!   W R O N G  result   !!, should be " << L <<"\n";
  278:     }
  279:     cout << endl;
  280:
  281: // Timings  and Performance
  282:     cout << endl;
  283:     cout.precision(2);
  284:
  285:
  286:     double Gflops = (2.0*L*N*M) / t_diff / 1024 / 1024 / 1024;
  287:     double MemBandwidth = (2.0*L*N*M + M*N)/ t_diff / 1024 / 1024 / 1024 * sizeof(A[
0]);
  288:
  289:     cout << "Total duration : " << t_diff*NLOOPS << endl;
  290:     cout << "Timing in sec. : " << t_diff << endl;
  291:     cout << "GFLOPS         : " << Gflops << endl;
  292:     cout << "GiByte/s       : " << MemBandwidth << endl;
  293:
  294:
  295:
  296:     return vector<double>{t_diff, Gflops, MemBandwidth};
  297: }
  298:
  299:
  300: vector<double> test_D(const size_t &NLOOPS, const size_t &N, const size_t &p)
  301: {
  302:     cout << "#################### (D) ####################" << endl;
  303:     cout << "\nLOOPS = " << NLOOPS << endl;
  304:     cout << "\nN = " << N << endl;
  305:     cout << "\np = " << p << endl;
  306:
  307: //  Memory allocation
  308:     cout << "Memory allocation\n";
  309:
  310:     vector<double> a(p + 1, 0);
  311:     vector<double> x(N);
  312:
  313:     cout.precision(2);
  314:     cout << (1.0*(p + 1) + N) *sizeof(x[0]) / 1024 / 1024 / 1024 << " GByte Memory a
llocated\n";
  315:     cout.precision(6);
  316:
  317: //  Data initialization
  318:
  319:     for (size_t j = 0; j < N; ++j)
  320:         x[j] = 1.0*j;
```

```
  321:
  322:     for (size_t k = 0; k < p + 1; ++k)
  323:         a[k] = pow(-1.0, k);          // poly(x) = 1 - x + x^2 - x^3 + x^4 - ...
  324:
  325:
  326:
  327:     cout << "\nStart Benchmarking poly\n";
  328:
  329:     auto t1 = system_clock::now(); // start timer
  330: // Do calculation
  331:     vector<double> y(N);
  332:     double check;
  333:     double check_sum;
  334:
  335:     for (size_t i = 0; i < NLOOPS; ++i)
  336:     {
  337:         y = poly_parallel(a, x);
  338:         check = y[0];
  339:
  340:         check_sum += check; // prevents the optimizer from removing unused calculati
on results.
  341:     }
  342:
  343:     auto t2 = system_clock::now();  // stop timer
  344:     auto duration = duration_cast<microseconds>(t2 - t1);         // duration in micr
oseconds
  345:     double t_diff = static_cast<double>(duration.count()) / 1e6; // overall duration
 in seconds
  346:     t_diff = t_diff/NLOOPS;                                      // duration per loo
p seconds
  347:
  348:
  349:
  350: // Check the correct result
  351:     cout << "\n poly(" << x[0] << ") = " << check << endl;
  352:     if (abs(check - 1.0) > 1.0/1e6)
  353:     {
  354:         cout << "  !!   W R O N G  result   !!\n";
  355:     }
  356:     cout << endl;
  357:
  358:
  359: // Timings  and Performance
  360:     cout << endl;
  361:     cout.precision(2);
  362:
  363:
  364:     double Gflops = (N*(p + 1)*3.0) / t_diff / 1024 / 1024 / 1024;
  365:     double MemBandwidth = (N*(2.0 + 3.0*(p + 1)))/ t_diff / 1024 / 1024 / 1024 * siz
eof(x[0]);
  366:
  367:     cout << "Total duration : " << t_diff*NLOOPS << endl;
  368:     cout << "Timing in sec. : " << t_diff << endl;
  369:     cout << "GFLOPS         : " << Gflops << endl;
  370:     cout << "GiByte/s       : " << MemBandwidth << endl;
  371:
  372:
  373:
  374:     return vector<double>{t_diff, Gflops, MemBandwidth};
  375: }
```

```
    1: #pragma once
    2: #include <vector>
    3: using namespace std;
    4:
    5: vector<double> test_A(const size_t &NLOOPS, const size_t &N);
    6:
    7: vector<double> test_A_sum(const size_t &NLOOPS, const size_t &N);
    8:
    9: vector<double> test_B(const size_t &NLOOPS, const size_t &N, const size_t &M);
   10:
   11: vector<double> test_C(const size_t &NLOOPS, const size_t &L, const size_t &M, const
size_t &N);
   12:
   13: vector<double> test_D(const size_t &NLOOPS, const size_t &N, const size_t &p);
```

```cpp
#include "benchmark_tests.h"
#include <iostream>
#include <cmath>

int main()
{
    vector<vector<double>> results_scalar;
    results_scalar.push_back(test_A(2000000, pow(10,3)));
    results_scalar.push_back(test_A(1000000, pow(10,4)));
    results_scalar.push_back(test_A(100000, pow(10,5)));
    results_scalar.push_back(test_A(10000, pow(10,6)));
    results_scalar.push_back(test_A(750, pow(10,7)));
    results_scalar.push_back(test_A(125, pow(10,8)));


    vector<vector<double>> results_sum;
    results_sum.push_back(test_A_sum(3000000, pow(10,3)));
    results_sum.push_back(test_A_sum(2000000, pow(10,4)));
    results_sum.push_back(test_A_sum(1000000, pow(10,5)));
    results_sum.push_back(test_A_sum(50000, pow(10,6)));
    results_sum.push_back(test_A_sum(2000, pow(10,7)));
    results_sum.push_back(test_A_sum(250, pow(10,8)));


    test_B(100, 20000, 10000);

    test_C(25, 500, 1000, 1500);

    test_D(100, 100, 1000000);



    cout << endl << "###### Scalar ######" << endl;
    cout << "Timing\tGFLOPS\tGiByte/s" << endl;
    cout << "-----------------------------" << endl;
    for (size_t i = 0; i < results_scalar.size(); ++i)
        cout << results_scalar[i][0] << "\t" << results_scalar[i][1] << "\t" << resu
lts_scalar[i][2] << endl;

    cout << endl << "###### Sum ######" << endl;
    cout << "Timing\tGFLOPS\tGiByte/s" << endl;
    cout << "-----------------------------" << endl;
    for (size_t i = 0; i < results_sum.size(); ++i)
        cout << results_sum[i][0] << "\t" << results_sum[i][1] << "\t" << results_su
m[i][2] << endl;



    // ###### Scalar ######
    // Timing  GFLOPS  GiByte/s
    // -----------------------------
    // 3.4e-06 0.54    4.3
    // 4.6e-06 4       32
    // 1.6e-05 12      95
    // 0.0011  1.7     13
    // 0.0097  1.9     15
    // 0.075   2.5     20


    // ###### Sum ######
    // Timing  GFLOPS  GiByte/s
    // -----------------------------
    // 5.5e-06 0.17    1.3
    // 5.4e-06 1.7     14
    // 1.5e-05 6.1     49
    // 0.00013 7.2     57
    // 0.0033  2.8     23
```

```
67:     // 0.032   2.9     23
68:
69:
70:
71:
72:
73:     // ######### NOT PARALLEL (from exercise sheet 2) #########
74:     //      Timing  GFLOPS  GiByte/s
75:     // ---------------------------------
76:     // (A)  0.038   2.5     20
77:     // (B)  0.13    2.9     23
78:     // (C)  0.44    3.2     25
79:     // (D)  0.19    1.5     12
80:
81:
82:
83:     return 0;
84: }
```