Семинар 2 (24)
Программирование сопроцессора Intel Xeon Phi

Михаил Курносов

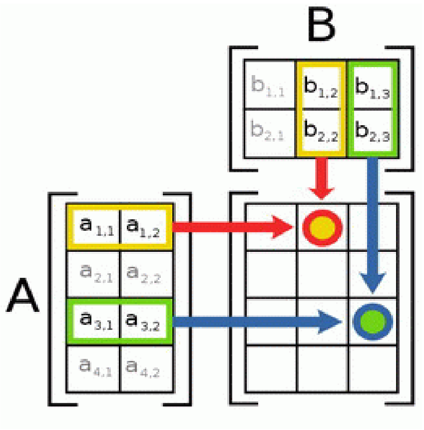
E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Цикл семинаров «Основы параллельного программирования»
Институт физики полупроводников им. А. В. Ржанова СО РАН
Новосибирск, 2016

Умножение матриц SGEMM (serial)

```
enum {
    N = 1000, M = 1000, Q = 1000,
    NREPS = 10,
};

/* Naive matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_host(float *a, float *b, float *c, int n, int m, int q)
{
    /* FP ops: 2 * n * q * m */
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < q; j++) {
            float s = 0.0;
            for (int k = 0; k < m; k++)
                s += a[i * m + k] * b[k * q + j];
            c[i * q + j] = s;
        }
    }
}
```

GigaFLOP = 2 * M * Q * N / 10^9



Умножение матриц SGEMM (serial)

```
double run_host(const char *msg, void (*sgemm_fun)(float *, float *, float *, int, int, int))
{
    double gflop = 2.0 * N * Q * M * 1E-9;
    float *a, *b, *c;
    a = malloc(sizeof(*a) * N * M);
    b = malloc(sizeof(*b) * M * Q);
    c = malloc(sizeof(*c) * N * Q);

    srand(0);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++)
            a[i * M + j] = rand() % 100;
    }
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < Q; j++)
            b[i * Q + j] = rand() % 100;
    }

    /* Warmup */
    double twarmup = wtime();
    sgemm_fun(a, b, c, N, M, Q);
    twarmup = wtime() - twarmup;
```

Умножение матриц SGEMM (serial)

```
    /* Measures */
    double tavg = 0.0;
    double tmin = 1E6;
    double tmax = 0.0;

    for (int i = 0; i < NREPS; i++) {
        double t = wtime();
        sgemm_fun(a, b, c, N, M, Q);
        t = wtime() - t;
        tavg += t;
        tmin = (tmin > t) ? t : tmin;
        tmax = (tmax < t) ? t : tmax;
    }
    tavg /= NREPS;
    printf("%s (%d runs): perf %.2f GFLOPS; time: tavg %.6f, tmin %.6f, tmax %.6f, twarmup %.6f\n",
           msg, NREPS, gflop / tavg, tavg, tmin, tmax, twarmup);

    free(c); free(b);  free(a);
    return tavg;
}
```

```
# cnmic Intel Xeon CPU E5-2620 v3
SGEMM N = 1000, M = 1000, Q = 1000
Host serial (10 runs): perf 1.80 GFLOPS; time: tavg 1.109740, tmin 1.108881, tmax 1.110844, twarmup 1.111684

SGEMM N = 2000, M = 2000, Q = 2000
Host serial (10 runs): perf 1.54 GFLOPS; time: tavg 10.358897, tmin 10.332893, tmax 10.547114, twarmup 10.571816
```

Умножение матриц SGEMM (serial): opt

```c
/* Matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_host_opt(float *a, float *b, float *c, int n, int m, int q)
{
    /* Permute loops k and j for improving cache utilization */
    for (int i = 0; i < n * q; i++)
        c[i] = 0;

    /* FP ops: 2 * n * m * q */
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < m; k++) {
            for (int j = 0; j < q; j++)
                c[i * q + j] += a[i * m + k] * b[k * q + j];
        }
    }
}
```

Умножение матриц SGEMM (OpenMP)

```
/* Matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_host_omp(float *a, float *b, float *c, int n, int m, int q)
{
    #pragma omp parallel
    {
        int k = 0;
        #pragma omp for
        for (int i = 0; i < n; i++)
            for (int j = 0; j < q; j++)
                c[k++] = 0.0;

        #pragma omp for
        for (int i = 0; i < n; i++) {
            for (int k = 0; k < m; k++) {
                for (int j = 0; j < q; j++)
                    c[i * q + j] += a[i * m + k] * b[k * q + j];
            }
        }
    }
}
```

Умножение матриц SGEMM (OpenMP)

```c
int main(int argc, char **argv)
{
    int omp_only = (argc > 1) ? 1 : 0;

    printf("SGEMM N = %d, M = %d, Q = %d\n", N, M, Q);
    if (!omp_only) {
        double t_host = run_host("Host serial", &sgemm_host);
        double t_host_opt = run_host("Host opt", &sgemm_host_opt);
        double t_host_omp = run_host("Host OMP", &sgemm_host_omp);

        printf("Speedup (host/host_opt): %.2f\n", t_host / t_host_opt);
        printf("Speedup (host_opt/host_OMP): %.2f\n", t_host_opt / t_host_omp);
    } else {
        char buf[256];
        sprintf(buf, "Host OMP %d", omp_get_max_threads());
        run_host(buf, &sgemm_host_omp);
    }
    return 0;
}
```

Умножение матриц SGEMM (OpenMP)

```c
int main(int argc, char **argv)
{
    int omp_only = (argc > 1) ? 1 : 0;

    printf("SGEMM N = %d, M = %d, Q = %d\n", N, M, Q);
    if (!omp_only) {
        double t_host = run_host("Host serial", &sgemm_host);
        double t_host_opt = run_host("Host opt", &sgemm_host_opt);
        double t_host_omp = run_host("Host OMP", &sgemm_host_omp);

}
```

```
# cnmic Intel Xeon CPU E5-2620 v3 (12 threads)
SGEMM N = 1000, M = 1000, Q = 1000
Host serial (10 runs): perf 1.80 GFLOPS; time: tavg 1.110385, tmin 1.109763, tmax 1.110921, twarmup 1.112136
Host opt (10 runs): perf 8.78 GFLOPS; time: tavg 0.227882, tmin 0.227810, tmax 0.228015, twarmup 0.228679
Host OMP (10 runs): perf 104.36 GFLOPS; time: tavg 0.019164, tmin 0.019143, tmax 0.019208, twarmup 0.036171
Speedup (host/host_opt): 4.87
Speedup (host_opt/host_OMP): 11.89

SGEMM N = 2000, M = 2000, Q = 2000
Host serial (10 runs): perf 1.60 GFLOPS; time: tavg 9.983190, tmin 9.972556, tmax 9.988208, twarmup 9.993524
Host opt (10 runs): perf 6.58 GFLOPS; time: tavg 2.429791, tmin 2.428237, tmax 2.430672, twarmup 2.432724
Host OMP (10 runs): perf 43.00 GFLOPS; time: tavg 0.372085, tmin 0.369906, tmax 0.379422, twarmup 0.384724
Speedup (host/host_opt): 4.11
Speedup (host_opt/host_OMP): 6.53
```

Умножение матриц SGEMM (Xeon Phi)

```c
/* Matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_phi(float *a, float *b, float *c, int n, int m, int q)
{
    #pragma offload target(mic) in(a:length(n * m)) in(b:length(m * q)) out(c:length(n * q))
    {
        #pragma omp parallel
        {
            int k = 0;
            #pragma omp for
            for (int i = 0; i < n; i++)
                for (int j = 0; j < q; j++)
                    c[k++] = 0.0;

            #pragma omp for
            for (int i = 0; i < n; i++) {
                for (int k = 0; k < m; k++) {
                    for (int j = 0; j < q; j++)
                        c[i * q + j] += a[i * m + k] * b[k * q + j];
                }
            }
        }
    }
}
```

Умножение матриц SGEMM (Xeon Phi)

```
/* Matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_phi(float *a, float *b, float *c, int n, int m, int q)
{
    #pragma offload target(mic) in(a:length(n * m)) in(b:length(m * q)) out(c:length(n * q))
    {
        #pragma omp parallel
        {
            int k = 0;
            #pragma omp for
            for (int i = 0; i < n; i++)
                for (int j = 0; j < q; j++)
                    c[k++] = 0.0;

            #pragma omp for
```

N = M = Q = 1000

```
# Intel Xeon Phi 3120A
SGEMM N = 1000, M = 1000, Q = 1000
Phi OMP 56 (5 runs): perf 31.49 GFLOPS; time: tavg 0.063517, tmin 0.060203, tmax 0.066313, twarmup 0.385153

SGEMM N = 1000, M = 1000, Q = 1000
Phi OMP 112 (5 runs): perf 40.44 GFLOPS; time: tavg 0.049456, tmin 0.045439, tmax 0.060661, twarmup 0.443696

SGEMM N = 1000, M = 1000, Q = 1000
Phi OMP 224 (5 runs): perf 39.34 GFLOPS; time: tavg 0.050835, tmin 0.047532, tmax 0.056592, twarmup 0.555559
```

```
}
```

Умножение матриц SGEMM (Xeon Phi)

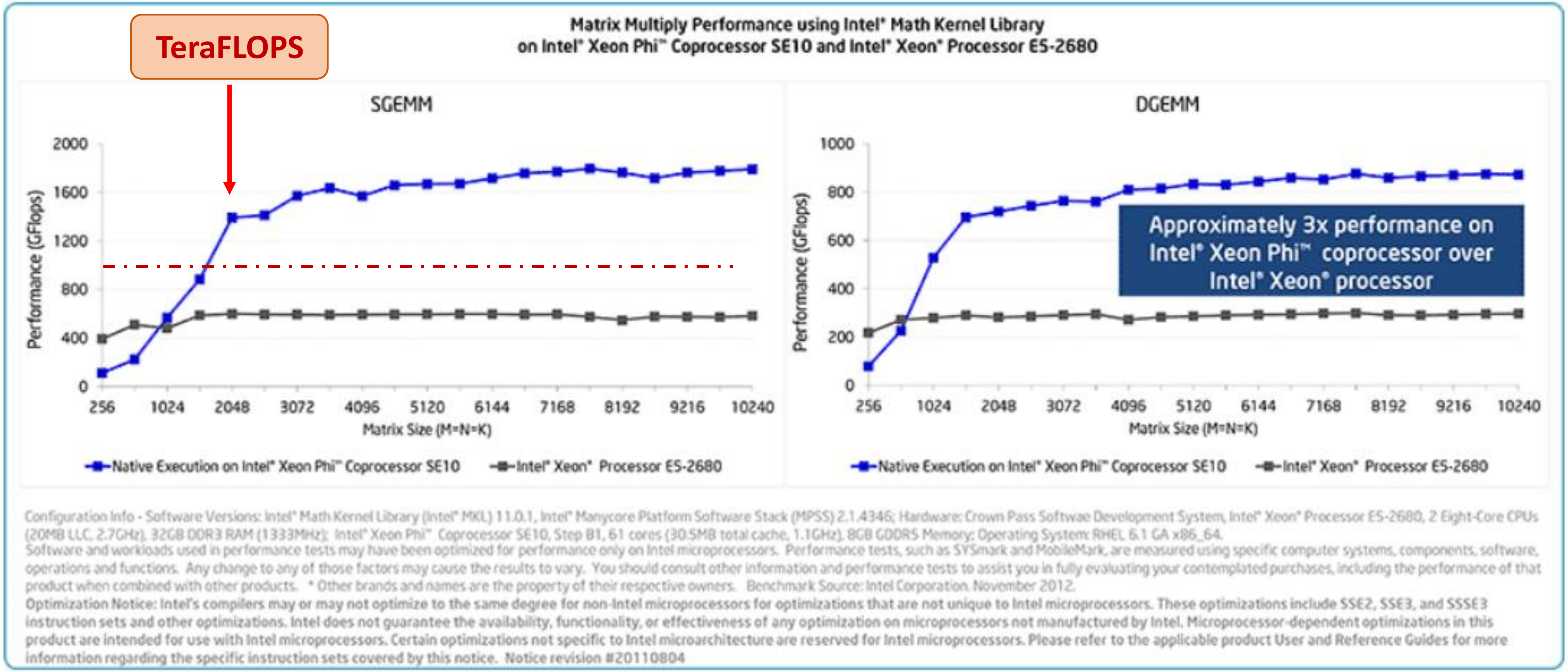
```
/* Matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_phi(float *a, float *b, float *c, int n, int m, int q)
{
    #pragma offload target(mic) in(a:length(n * m)) in(b:length(m * q)) out(c:length(n * q))
    {
        #pragma omp parallel
        {
            int k = 0;
            #pragma omp for
            for (int i = 0; i < n; i++)
                for (int j = 0; j < q; j++)
                    c[k++] = 0.0;

            #pragma omp for
```

N = M = Q = 5000

```
# Intel Xeon Phi 3120A
SGEMM N = 5000, M = 5000, Q = 5000
Phi OMP 112 (5 runs): perf 75.76 GFLOPS; time: tavg 3.299893, tmin 3.273967, tmax 3.379286, twarmup 4.431175

SGEMM N = 5000, M = 5000, Q = 5000
Phi OMP 224 (5 runs): perf 82.86 GFLOPS; time: tavg 3.017069, tmin 2.916298, tmax 3.143453, twarmup 4.696685
```

```
        }
    }
}
```

Умножение матриц SGEMM (Intel MKL)



Умножение матриц SGEMM (Intel MKL)

```c
#include <mkl.h>    // for sgemm

/* Matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_phi_mkl(float *a, float *b, float *c, int n, int m, int q)
{
    /*
     * sblas_sgemm: C[] = alpha * A[] x B[] + beta * C[]
     */
    float alpha = 1.0;
    float beta = 0.0;
    #pragma offload target(mic) in(a:length(n * m)) in(b:length(m * q)) out(c:length(n * q))
    {
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, n, q, m, alpha, a, m, b, q, beta, c, q);
    }
}


double run_phi(const char *msg, void (*sgemm_fun)(float *, float *, float *, int, int, int))
{
    a = mkl_malloc(sizeof(*a) * N * M, 64);
    // ...

    mkl_free(a);
    return tavg;
}
```

```makefile
# Makefile
CFLAGS := -Wall -g -std=c99 -fopenmp -mkl -O3
LDFLAGS := -mkl -lm -fopenmp
```

Умножение матриц SGEMM (Intel MKL)

```
#include <mkl.h>    // for sgemm

/* Matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_phi_mkl(float *a, float *b, float *c, int n, int m, int q)
{
    /*
     * sblas_sgemm: C[] = alpha * A[] x B[] + beta * C[]
     */
    float alpha = 1.0;
    float beta = 0.0;
    #pragma offload target(mic) in(a:length(n * m)) in(b:length(m * q)) out(c:length(n * q))
    {
        cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, n, q, m, alpha, a, m, b, q, beta, c, q);
```

```
# Intel Xeon Phi 3120A
SGEMM N = 1000, M = 1000, Q = 1000
Phi MKL 224 (10 runs): perf 72.16 GFLOPS; time: tavg 0.027717, tmin 0.024476, tmax 0.042179, twarmup 1.352913

SGEMM N = 5000, M = 5000, Q = 5000
Phi MKL 224 (10 runs): perf 375.38 GFLOPS; time: tavg 0.665999, tmin 0.650047, tmax 0.773388, twarmup 2.701640

SGEMM N = 10000, M = 10000, Q = 10000
Phi MKL 224 (10 runs): perf 525.74 GFLOPS; time: tavg 3.804181, tmin 3.762794, tmax 3.946551, twarmup 8.159131

SGEMM N = 15000, M = 15000, Q = 15000
Phi MKL 224 (10 runs): perf 492.96 GFLOPS; time: tavg 13.692805, tmin 13.589008, tmax 14.125500, twarmup 22.575884
```

Умножение матриц SGEMM (Intel MKL)

```
# launch.sh
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=672

export MIC_USE_2MB_BUFFERS=10M

export MIC_KMP_AFFINITY=explicit,granularity=fine,proclist=[1-224:1]
./sgemm
```

The Intel compiler offload runtime allocates memory with 2MB pages when the size of allocation exceeds the value of the MIC_USE_2MB_BUFFERS environment variable

```
# Intel Xeon Phi 3120A
SGEMM N = 10000, M = 10000, Q = 10000
Phi MKL 672 (5 runs): perf 929.12 GFLOPS; time: tavg 2.152566, tmin 2.141921, tmax 2.167960, twarmup 4.338107

SGEMM N = 15000, M = 15000, Q = 15000
Phi MKL 672 (5 runs): perf 929.59 GFLOPS; time: tavg 7.261260, tmin 7.253318, tmax 7.270383, twarmup 10.653565

SGEMM N = 20000, M = 20000, Q = 20000
Phi MKL 672 (5 runs): perf 1237.86 GFLOPS; time: tavg 12.925547, tmin 12.906175, tmax 12.946699, twarmup 18.077353
```

1.2 TeraFLOPS

- How to Use Huge Pages to Improve Application Performance on Intel® Xeon Phi™ Coprocessor // https://software.intel.com/sites/default/files/Large_pages_mic_0.pdf
- http://www.intuit.ru/studies/professional_skill_improvements/17248/courses/1096/lecture/22919?page=2

Умножение матриц SGEMM (Xeon Phi)

```
/* Matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_phi(float *a, float *b, float *c, int n, int m, int q)
{
    #pragma offload target(mic) in(a:length(n * m)) in(b:length(m * q)) out(c:length(n * q))
    {
        #pragma omp parallel
        {
            int k = 0;
            #pragma omp for
            for (int i = 0; i < n; i++)
                for (int j = 0; j < q; j++)
                    c[k++] = 0.0;

            #pragma omp for
```

```
# Intel Xeon Phi 3120A + MIC_USE_2MB_BUFFERS=10M + thread affinity
GEMM N = 10000, M = 10000, Q = 10000
Phi OMP 672 (3 runs): perf 75.20 GFLOPS; time: tavg 26.597019, tmin 25.990380, tmax 26.983693, twarmup 28.261966
```

```
                }
            }
        }
    }
}
```

PCI Express bandwidth test

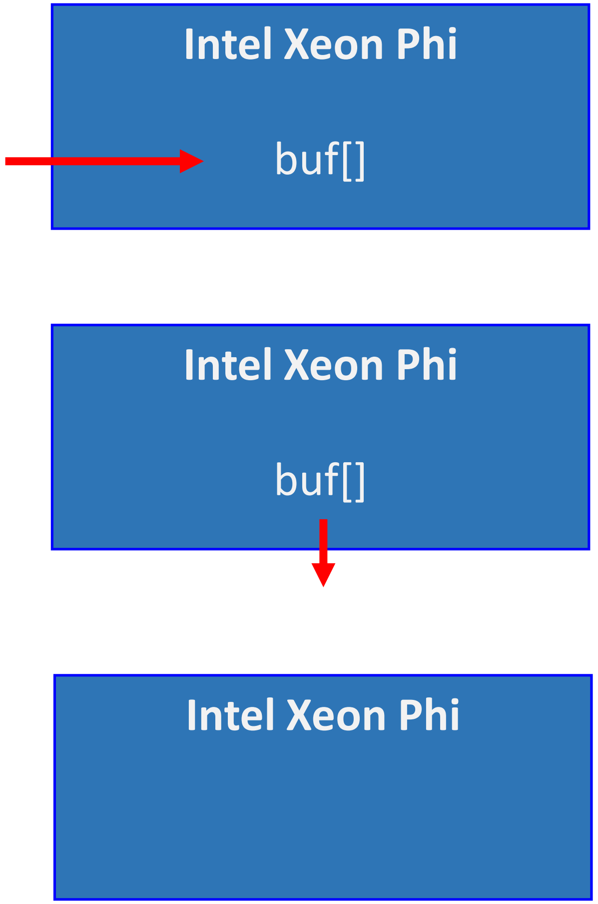
```c
int main(int argc, char **argv)
{
    printf("Xeon Phi bwtest: nreps = %d\n", NREPS);
    printf("size        alignment  talloc  tsend  trecv\n");

    int s[] = {2, 4, 8, 16, 32, 64, 128, 256, 512, 1024};
    int a[] = {32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384};

    for (int j = 0; j < NELEMS(a); j++) {
        for (int i = 0; i < NELEMS(s); i++) {
            testbw(s[i] * 1024 * 1024, a[j]);
        }
    }
    return 0;
}
```

PCI Express bandwidth test

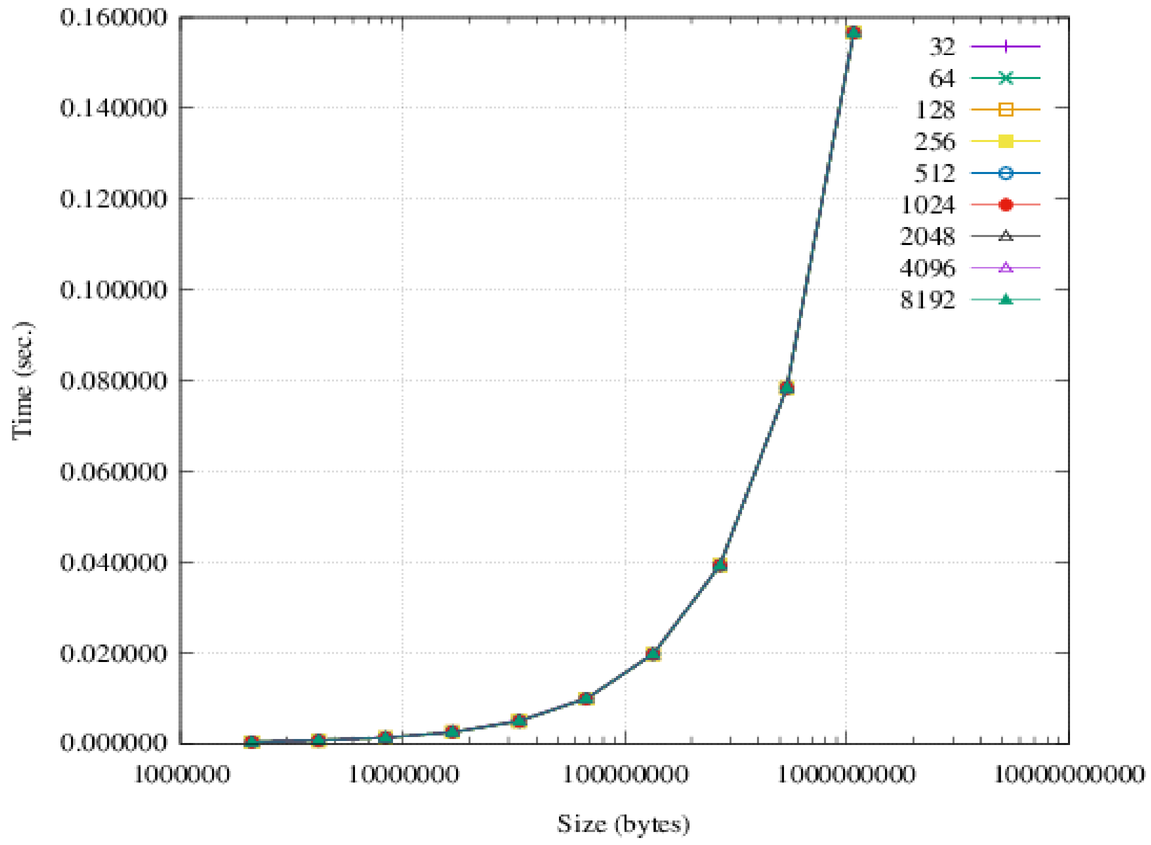
```c
void testbw(int size, int alignment)
{
    uint8_t *buf = _mm_malloc(size, alignment);
    if (!buf) {
        fprintf(stderr, "Can't allocate memory\n");
        exit(EXIT_FAILURE);
    }

    // Init buffer (allocate pages)
    memset(buf, 1, size);
    double t, talloc;
    double tsend = 0.0;
    double trecv = 0.0;

    // Allocate buffer on Phi
    talloc = wtime();
    #pragma offload target(mic) in(buf:length(size) free_if(0))
    { }
    talloc = wtime() - talloc;
```

PCI Express bandwidth test

```
    // Measures
    for (int i = 0; i < NREPS; i++) {
        // Copy to Phi
        t = wtime();
        #pragma offload target(mic) in(buf:length(size) alloc_if(0) free_if(0))
        { }
        tsend += wtime() - t;

        // Copy from Phi
        t = wtime();
        #pragma offload target(mic) out(buf:length(size) alloc_if(0) free_if(0))
        { }
        trecv += wtime() - t;
    }

    // Free on Phi
    #pragma offload target(mic) in(buf:length(size) alloc_if(0) free_if(1))
    { }

    tsend /= NREPS;
    trecv /= NREPS;

    printf("%-10d  %-8d  %-.6f  %-.6f  %-.6f\n", size, alignment, talloc, tsend, trecv);
    _mm_free(buf);
}
```



PCI Express bandwidth test



Задание

- Реализуйте умножение матриц для типа double (dgemm)
- Сравните производительность с версией из Intel MKL (cblas_dgemm)