Лекция 8
Введение в программирование сопроцессора Intel Xeon Phi

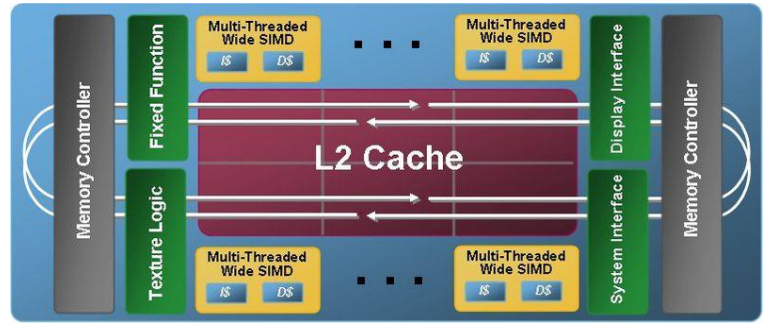
Курносов Михаил Георгиевич

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

Курс «Параллельное программирование»
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)
Осенний семестр, 2017

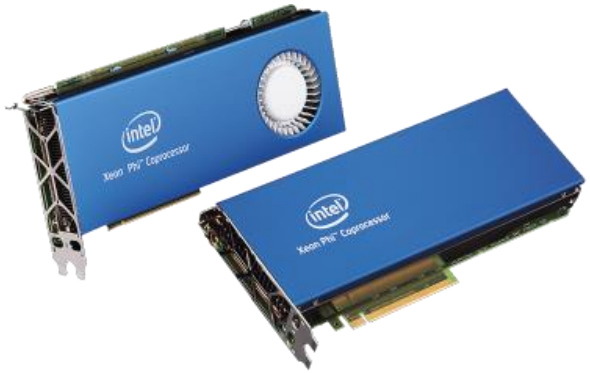
Intel Xeon Phi

- **Intel Teraflops Research Chip** (Polaris, 200x-2007 гг.)
  80 cores, 80 routers, self-correction system
- **Single-Chip Cloud Computer (SCC, 2009)**
  48 P54C Pentium cores, 4x6 2D-mesh of tiles (2 cores)
- **Intel Larrabee (200X-2010 гг.)**
  GPGPU: x86, cache coherency, 1024-bit ring bus, 4-way SMT,
  разработка отменена в 2010 г.



Larrabee GPU
(SIGGRAPH, 2008)

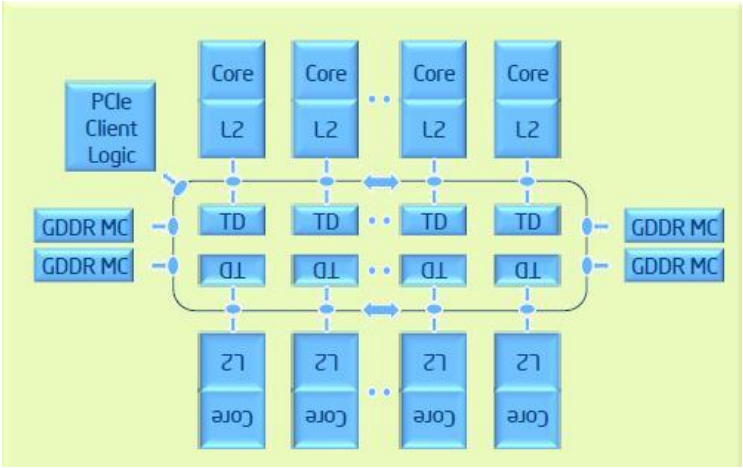
- **Intel MIC (Intel Many Integrated Core Architecture)**
  кеш-когерентный мультипроцессор с общей памятью,
  аппаратная многопоточность (4-way SMT),
  широкие векторные регистры (512 бит)



  - **Knights Ferry** (2010): 32 in-order cores, 4-way SMT, ring bus, 750 GFLOPS
  - **Knights Corner/ Xeon Phi** (2011): 22 nm, >= 50 cores
  - **Knights Landing** (2nd gen., 14 nm, 2013): Silvermont (Atom) cores, 4 threads per core
  - **Knights Hill** (3rd gen., 10 nm)

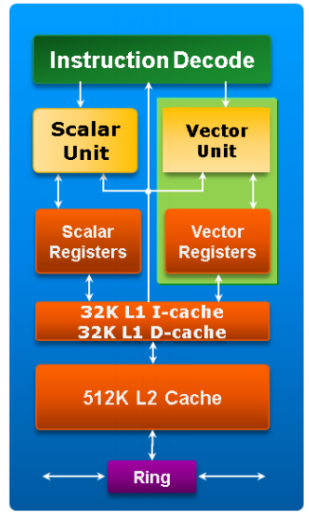
Intel Knights Corner micro-architecture

- **Больше 50 ядер Pentium P54C**:
  - x86 ISA, in-order, 4-way SMT, 512-bit SIMD units
  - Cache: 32 KB L1 data/i-cache, coherent L2 cache (512 KB), двусторонняя кольцевая шина
- Кольцевая шина: двусторонняя
- Подключается через шину PCI Express



- **cnmic (oak): Intel Xeon Phi 3120A**
  - **Cores:** 57 4-way SMT (core #57 for OS only!): 224 threads
  - **RAM**: GDDR5 6 GiB
  - **Intel Compiler:**

```
$ source /opt/intel_cluster/bin/iccvars.sh intel64
```



https://software.intel.com/sites/default/files/managed/ee/4e/intel-xeon-phi-coprocessor-quick-start-developers-guide.pdf

https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner

Подсчет простых чисел (serial version)

```
int is_prime_number(int n)
{
    int limit = sqrt(n) + 1;
    for (int i = 2; i <= limit; i++) {
        if (n % i == 0) return 0;
    }
    return (n > 1) ? 1 : 0;
}

int count_prime_numbers(int a, int b)
{
    int nprimes = 0;
    if (a <= 2) {           /* Count '2' as a prime number */
        nprimes = 1;
        a = 2;
    }
    if (a % 2 == 0)         /* Shift 'a' to odd number */
        a++;

    /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
    for (int i = a; i <= b; i += 2) {
        if (is_prime_number(i))
            nprimes++;
    }
    return nprimes;
}
```

Число операций $O(\sqrt{n})$

Проверка всех нечетных чисел в интервале $[a, b]$

Подсчет простых чисел (OpenMP)

```
int count_prime_numbers_omp(int a, int b)
{
    int nprimes = 0;
    /* Count '2' as a prime number */
    if (a <= 2) {
        nprimes = 1;
        a = 2;
    }
    /* Shift 'a' to odd number */
    if (a % 2 == 0)
        a++;

    #pragma omp parallel
    {
        /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
        #pragma omp for schedule(dynamic, 100) reduction(+:nprimes)
        for (int i = a; i <= b; i += 2) {
            if (is_prime_number(i))
                nprimes++;
        }
    }
    return nprimes;
}
```

Подсчет простых чисел (OpenMP)

```
int count_prime_numbers_omp(int a, int b)
{
    int nprimes = 0;
    /* Count '2' as a prime number */
    if (a <= 2) {
        nprimes = 1;
        a = 2;
    }
    /* Shift 'a' to odd number */
    if (a % 2 == 0)
        a++;

    #pragma omp parallel
    {
        /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
        #pragma omp for schedule(dynamic, 100) reduction(+:nprimes)
        for (int i = a; i <= b; i += 2) {
            if (is_prime_number(i))
                nprimes++;
        }
    }
    return nprimes;
}
```

cnmic (oak.cpct.sibsutis.ru)

System board: ASUS Z10PE-D8 WS (Dual CPU)
CPU: 2 x Intel Xeon E5-2620v3 (2.40GHz, Haswell, 6 cores)
RAM: 64 GiB, DDR4 2133 Mhz (4x8 GiB, 4x8 GiB)
Coprocessor: Intel Xeon Phi 3120A (Cores: 56 4-way SMT, RAM: GDDR5 6 GiB)

```
$ icc -std=c99 -Wall -O2 -fopenmp -c primes.c -o primes.o
$ icc -o primes primes.o -lm -fopenmp
```

```
$ export OMP_NUM_THREADS=12
$ ./primes
Count prime numbers in [1, 3000000]
Result (host serial): 216816
Result (host parallel): 216816
Execution time (host serial): 1.213376
Execution time (host parallel): 0.103723
Speedup host_serial/host_omp: 11.70
```

Подсчет простых чисел: Xeon Phi offload (serial)

```
#include <offload.h>

double run_phi_serial()
{
    #ifdef __INTEL_OFFLOAD
    printf("Intel Xeon Phi devices: %d\n", _Offload_number_of_devices());
    #endif

    int n;
    double t = wtime();
    #pragma offload target(mic) out(n)
    {
        n = count_prime_numbers_phi(a, b);
    }
    t = wtime() - t;

    printf("Result (phi serial): %d\n", n);
    return t;
}
```

Структурный блок *выгружается* (offload)
для выполнения на сопроцессор

Требуется указать:

- объекты, которые необходимо скопировать в память сопроцессора перед выполнением блока (**in**)
- объекты, которые необходимо скопировать из памяти сопроцессора после выполнением блока (**out**)

Подсчет простых чисел: Xeon Phi offload (serial)

```
__attribute__((target(mic))) int count_prime_numbers_phi(int a, int b)
{
    int nprimes = 0;

    /* Count '2' as a prime number */
    if (a <= 2) {
        nprimes = 1;
        a = 2;
    }

    /* Shift 'a' to odd number */
    if (a % 2 == 0)
        a++;

    /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
    for (int i = a; i <= b; i += 2) {
        if (is_prime_number(i))
            nprimes++;
    }
    return nprimes;
}
```

Функции и переменные, доступ к которым осуществляется на сопроцессоре, помечаются атрибутом

`__attribute__((target(mic)))`

(гетерогенная компиляция)

Подсчет простых чисел: Xeon Phi offload (serial)

```
__attribute__((target(mic))) int is_prime_number(int n)
{
    int limit = sqrt(n) + 1;
    for (int i = 2; i <= limit; i++) {
        if (n % i == 0)
            return 0;
    }
    return (n > 1) ? 1 : 0;
}
```

Функции и переменные, доступ к которым осуществляется на сопроцессоре, помечаются атрибутом

`__attribute__((target(mic)))`

(гетерогенная компиляция)

Подсчет простых чисел: Xeon Phi offload (serial)

```
int main(int argc, char **argv)
{
    printf("Count prime numbers in [%d, %d]\n", a, b);
    double thost_serial = run_host_serial();
    double thost_par = run_host_parallel();
    double tphi_serial = run_phi_serial();

    printf("Execution time (host serial): %.6f\n", thost_serial);
    printf("Execution time (host parallel): %.6f\n", thost_par);
    printf("Execution time (phi serial): %.6f\n", tphi_serial);
    printf("Ratio phi_serial/host_serial: %.2f\n", tphi_serial / thost_serial);
    printf("Speedup host_serial/host_omp: %.2f\n", thost_serial / thost_par);

    return 0;
}
```

```
$ export OMP_NUM_THREADS=12
$ ./primes
Count prime numbers in [1, 3000000]
Result (host serial): 216816
Result (host parallel): 216816
Intel Xeon Phi devices: 1
Result (phi serial): 216816
Execution time (host serial): 1.213118
Execution time (host parallel): 0.103338
Execution time (phi serial): 18.031396
Ratio phi_serial/host_serial: 14.86
Speedup host_serial/host_omp: 11.74
```

На данном тесте ядро Intel Xeon Phi 3120A
в ~15 раз медленнее ядра Intel Xeon E5-2620 v3

Подсчет простых чисел: Xeon Phi offload (OpenMP)

```
double run_phi_parallel()
{
    #ifdef __INTEL_OFFLOAD
    printf("Intel Xeon Phi devices: %d\n", _Offload_number_of_devices());
    #endif

    int n;
    double t = wtime();
    #pragma offload target(mic) out(n)
    n = count_prime_numbers_phi_omp(a, b);
    t = wtime() - t;

    printf("Result (phi parallel): %d\n", n);
    return t;
}
```

Подсчет простых чисел: Xeon Phi offload (OpenMP)

```
__attribute__((target(mic))) int count_prime_numbers_phi_omp(int a, int b)
{
    int nprimes = 0;
    /* Count '2' as a prime number */
    if (a <= 2) {
        nprimes = 1;
        a = 2;
    }
    /* Shift 'a' to odd number */
    if (a % 2 == 0)
        a++;

    /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
    #pragma omp parallel
    {
        #pragma omp for schedule(dynamic, 100) reduction(+:nprimes)
        for (int i = a; i <= b; i += 2) {
            if (is_prime_number(i))
                nprimes++;
        }
    }
    return nprimes;
}
```

Подсчет простых чисел: Xeon Phi offload (OpenMP)

```
int main(int argc, char **argv)
{
    printf("Count prime numbers in [%d, %d]\n", a, b);
    double thost_serial = run_host_serial();
    double thost_par = run_host_parallel();
    double tphi_serial = run_phi_serial();
    double tphi_par = run_phi_parallel();

    printf("Execution time (host serial): %.6f\n", thost_serial);
    printf("Execution time (host parallel): %.6f\n", thost_par);
    printf("Execution time (phi serial): %.6f\n", tphi_serial);
    printf("Execution time (phi parallel): %.6f\n", tphi_par);
    printf("Ratio phi_serial/host_serial: %.2f\n", tphi_serial / thost_serial);
    printf("Speedup host_serial/host_omp: %.2f\n", thost_serial / thost_par);
    printf("Speedup host_omp/phi_omp: %.2f\n", thost_par / tphi_par);
    printf("Speedup host_serial/phi_omp: %.2f\n", thost_serial / tphi_par);
    printf("Speedup phi_serial/phi_omp: %.2f\n", tphi_serial / tphi_par);

    return 0;
}
```

Подсчет простых чисел: Xeon Phi offload (OpenMP)

```
int main(int argc, char **argv)
{
    printf("Count prime numbers in [%d, %d]\n", a, b);
    double thost_serial = run_host_serial();
    double thost_par = run_host_parallel();
    double tphi_serial = run_phi_serial();
    double tphi_par = run_phi_parallel();

    printf("Execution time (host serial): %.6f\n", thost_serial);
    printf("Execution time (host parallel): %.6f\n", thost_par);
    printf("Execution time (phi serial): %.6f\n", tphi_serial);
    printf("Execution time (phi parallel): %.6f\n", tphi_par);
    printf("Ratio phi_serial/host_serial: %.2f\n", tphi_serial / thost_serial);
    printf("Speedup host_serial/host_omp: %.2f\n", thost_serial / thost_par);
    printf("Speedup host_omp/phi_omp: %.2f\n", thost_par / tphi_par);
    printf("Speedup host_serial/phi_omp: %.2f\n", thost_serial / tphi_pa
    printf("Speedup phi_serial/phi_omp: %.2f\n", tphi_serial / tphi_par);

    return 0;
}
```

```
$ cat ./launch.sh
#!/bin/sh

# Host OpenMP
export OMP_NUM_THREADS=12

# Xeon Phi OpenMP
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=224

./primes
```

```
$ ./launch.sh
Count prime numbers in [1, 3000000]
Execution time (host serial): 1.213278
Execution time (host parallel): 0.104636
Execution time (phi serial): 18.021162
Execution time (phi parallel): 0.770084
Ratio phi_serial/host_serial: 14.85
Speedup host_serial/host_omp: 11.60
Speedup host_omp/phi_omp: 0.14
Speedup host_serial/phi_omp: 1.58
Speedup phi_serial/phi_omp: 23.40
```

- Phi-OpenMP-версия на 224 потоках в 23 раза быстрее последовательной Phi-версии
- Phi-OpenMP-версия медленее Host-OpenMP-версии в ~7 раз

Xeon Phi: привязка потоков к ядрам (affinity)

```
$ cat ./launch.sh
#!/bin/sh

# Host OpenMP
export OMP_NUM_THREADS=12

# Xeon Phi OpenMP
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=56
export MIC_KMP_AFFINITY=verbose

./primes
```

```
Execution time (host serial): 1.213139
Execution time (host parallel): 0.108063
Execution time (phi serial): 18.027308
Execution time (phi parallel): 0.541261
Ratio phi_serial/host_serial: 14.86
Speedup host_serial/host_omp: 11.23
Speedup host_omp/phi_omp: 0.20
Speedup host_serial/phi_omp: 2.24
Speedup phi_serial/phi_omp: 33.31
```

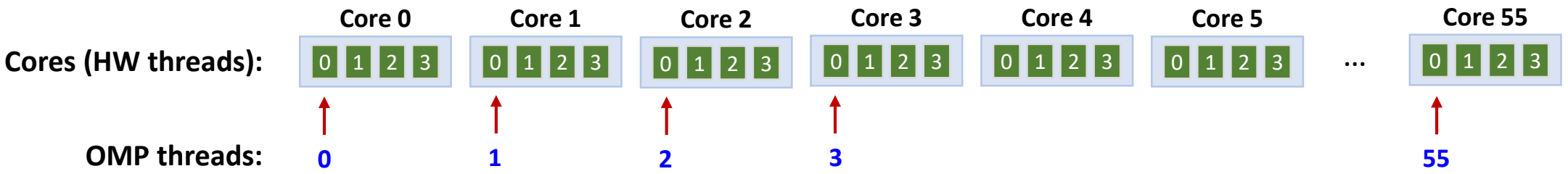
```
OMP: Info #204: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #205: KMP_AFFINITY: cpuid leaf 11 not supported - decoding legacy APIC ids.
OMP: Info #149: KMP_AFFINITY: Affinity capable, using global cpuid info
OMP: Info #154: KMP_AFFINITY: Initial OS proc set respected:
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44
,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85
,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,
120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150
,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,18
1,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,210,211,2
12,213,214,215,216,217,218,219,220,221,222,223,224}
OMP: Info #156: KMP_AFFINITY: 224 available OS procs
OMP: Info #157: KMP_AFFINITY: Uniform topology
OMP: Info #159: KMP_AFFINITY: 1 packages x 56 cores/pkg x 4 threads/core (56 total cores)
```

Xeon Phi: привязка потоков к ядрам (affinity)

```
OMP: Info #206: KMP_AFFINITY: OS proc to physical thread map:
OMP: Info #171: KMP_AFFINITY: OS proc 1 maps to package 0 core 0 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 2 maps to package 0 core 0 thread 1
OMP: Info #171: KMP_AFFINITY: OS proc 3 maps to package 0 core 0 thread 2
OMP: Info #171: KMP_AFFINITY: OS proc 4 maps to package 0 core 0 thread 3
OMP: Info #171: KMP_AFFINITY: OS proc 5 maps to package 0 core 1 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 6 maps to package 0 core 1 thread 1
OMP: Info #171: KMP_AFFINITY: OS proc 7 maps to package 0 core 1 thread 2
OMP: Info #171: KMP_AFFINITY: OS proc 8 maps to package 0 core 1 thread 3
OMP: Info #171: KMP_AFFINITY: OS proc 9 maps to package 0 core 2 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 10 maps to package 0 core 2 thread 1
OMP: Info #171: KMP_AFFINITY: OS proc 11 maps to package 0 core 2 thread 2
OMP: Info #171: KMP_AFFINITY: OS proc 12 maps to package 0 core 2 thread 3
...
OMP: Info #171: KMP_AFFINITY: OS proc 221 maps to package 0 core 55 thread 0
OMP: Info #171: KMP_AFFINITY: OS proc 222 maps to package 0 core 55 thread 1
OMP: Info #171: KMP_AFFINITY: OS proc 223 maps to package 0 core 55 thread 2
OMP: Info #171: KMP_AFFINITY: OS proc 224 maps to package 0 core 55 thread 3
```

Xeon Phi: привязка потоков к ядрам (affinity)

```
OMP: Info #242: KMP_AFFINITY: pid 12242 thread 0 bound to OS proc set {1}
OMP: Info #242: KMP_AFFINITY: pid 12242 thread 1 bound to OS proc set {5}
OMP: Info #242: KMP_AFFINITY: pid 12242 thread 2 bound to OS proc set {9}
OMP: Info #242: KMP_AFFINITY: pid 12242 thread 3 bound to OS proc set {13}
...
OMP: Info #242: KMP_AFFINITY: pid 12242 thread 54 bound to OS proc set {217}
OMP: Info #242: KMP_AFFINITY: pid 12242 thread 55 bound to OS proc set {221}
```



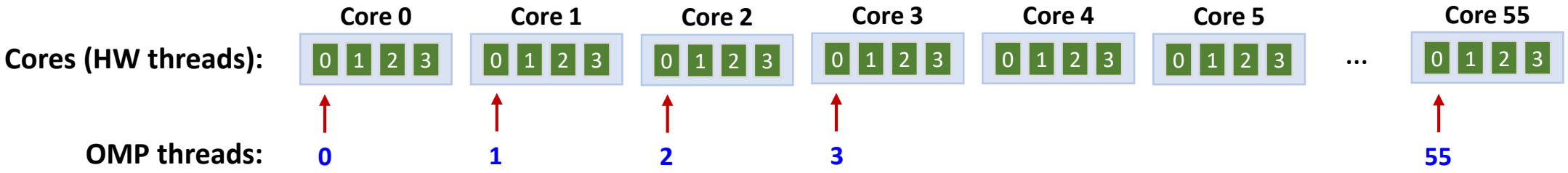
Default Affinity

Xeon Phi: привязка потоков к ядрам (affinity)

```
$ cat ./launch.sh
#!/bin/sh

# Host OpenMP
export OMP_NUM_THREADS=12

# Xeon Phi OpenMP
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=56
export MIC_KMP_AFFINITY=granularity=fine,balanced

./primes
```

```
Execution time (host serial): 1.213125
Execution time (host parallel): 0.103671
Execution time (phi serial): 18.028928
Execution time (phi parallel): 0.482872
Ratio phi_serial/host_serial: 14.86
Speedup host_serial/host_omp: 11.70
Speedup host_omp/phi_omp: 0.21
Speedup host_serial/phi_omp: 2.51
Speedup phi_serial/phi_omp: 37.34
```

fine, balanced

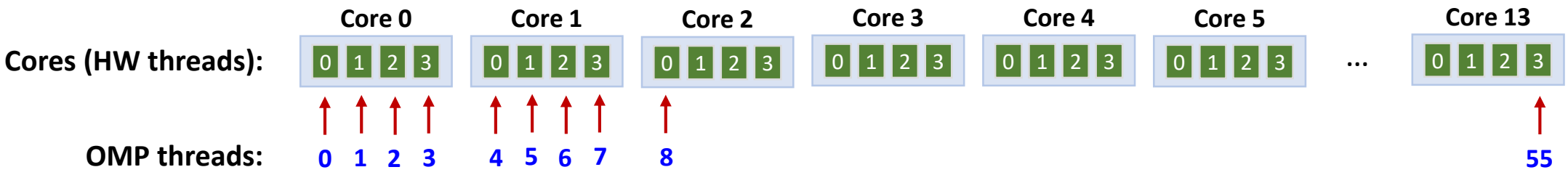


Xeon Phi: привязка потоков к ядрам (affinity)

```
$ cat ./launch.sh
#!/bin/sh

# Host OpenMP
export OMP_NUM_THREADS=12

# Xeon Phi OpenMP
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=56
export MIC_KMP_AFFINITY=granularity=fine,compact
./primes
```

```
Execution time (host serial): 1.213115
Execution time (host parallel): 0.103378
Execution time (phi serial): 18.030468
Execution time (phi parallel): 1.478703
Ratio phi_serial/host_serial: 14.86
Speedup host_serial/host_omp: 11.73
Speedup host_omp/phi_omp: 0.07
Speedup host_serial/phi_omp: 0.82
Speedup phi_serial/phi_omp: 12.19
```

fine, compact

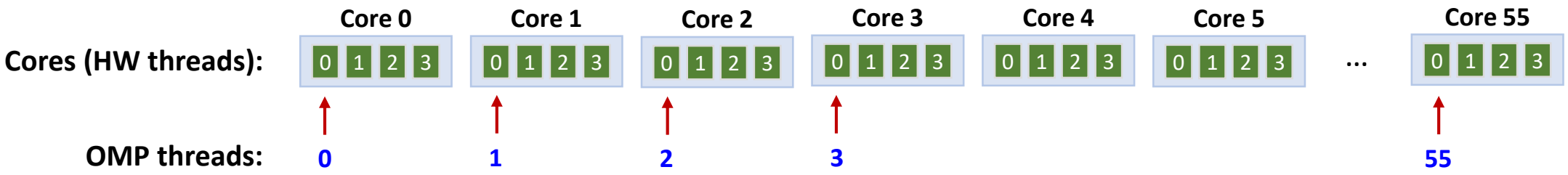


Xeon Phi: привязка потоков к ядрам (affinity)

```
$ cat ./launch.sh
#!/bin/sh

# Host OpenMP
export OMP_NUM_THREADS=12

# Xeon Phi OpenMP
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=56
export MIC_KMP_AFFINITY=granularity=fine,scatter
./primes
```

```
Execution time (host serial): 1.213140
Execution time (host parallel): 0.108781
Execution time (phi serial): 18.024953
Execution time (phi parallel): 0.551100
Ratio phi_serial/host_serial: 14.86
Speedup host_serial/host_omp: 11.15
Speedup host_omp/phi_omp: 0.20
Speedup host_serial/phi_omp: 2.20
Speedup phi_serial/phi_omp: 32.71
```

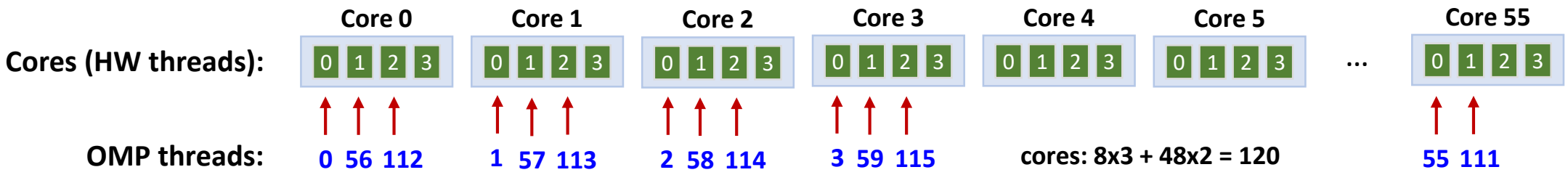
fine, scatter



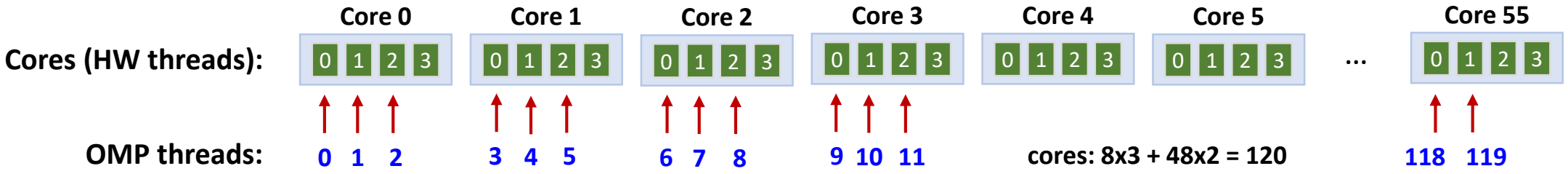
Xeon Phi: привязка потоков к ядрам (affinity)

NUM_THREADS = 120

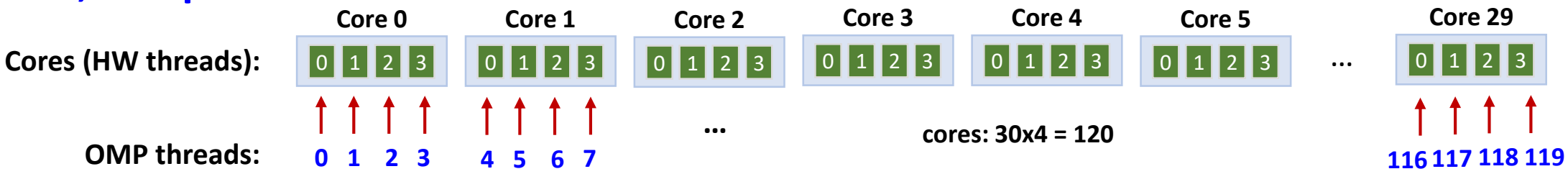
fine, scatter: core_id = thread_id % ncores



fine, balanced:



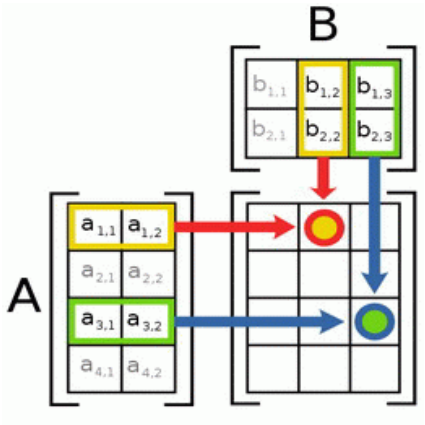
fine, compact:



Умножение матриц
SGEMM

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

```
/* 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)

```
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)

```
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)

```
/* 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)

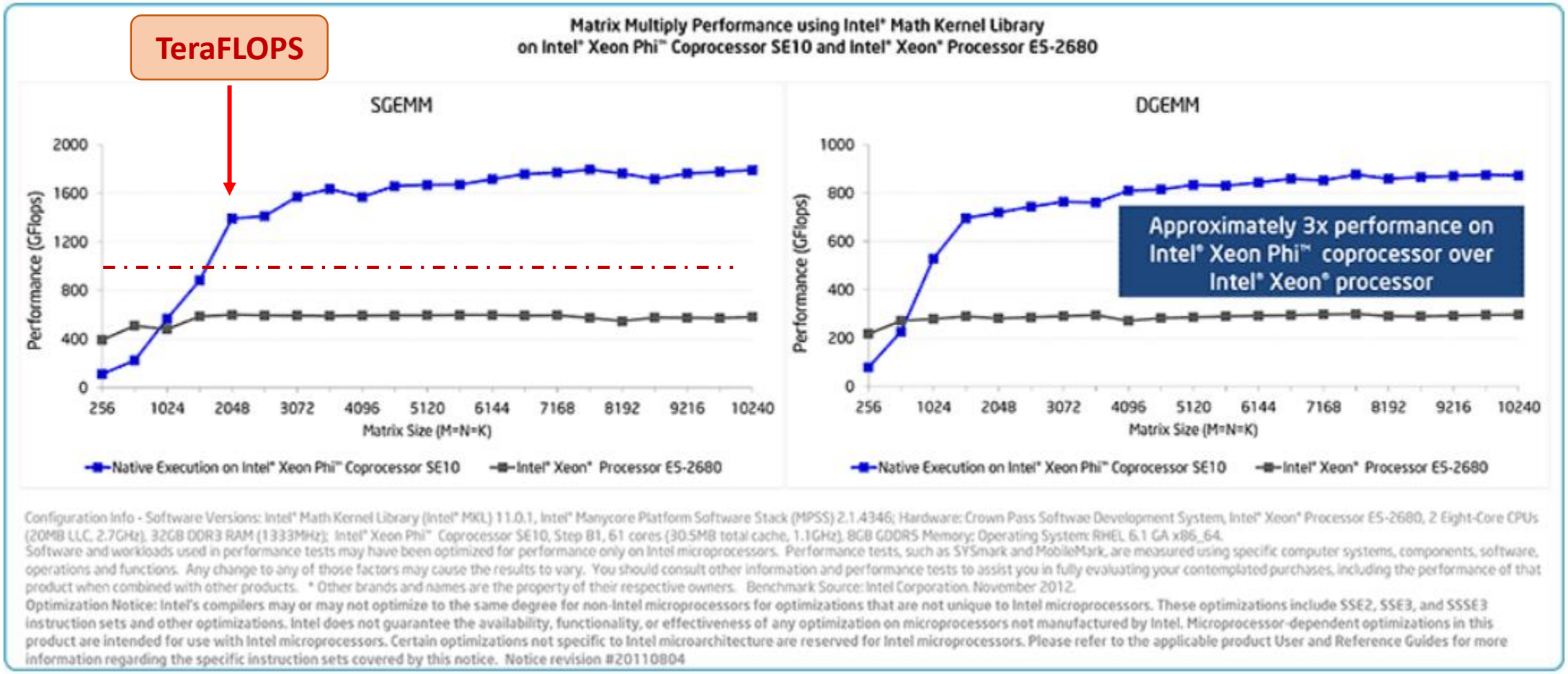
```
/* Matrix multiplication C[n, q] = A[n, m] * B[m, q] */
void sgemm_phi(float *a, float *b, float *c, int n, int m, int q)
{
    #pragma offload target(mic) in(a:length(n * m)) in(b:length(m * q)) out(c:length(n * q))
    {
        #pragma omp parallel
        {
            int k = 0;
           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

```c
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
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

PCI Express bandwidth test

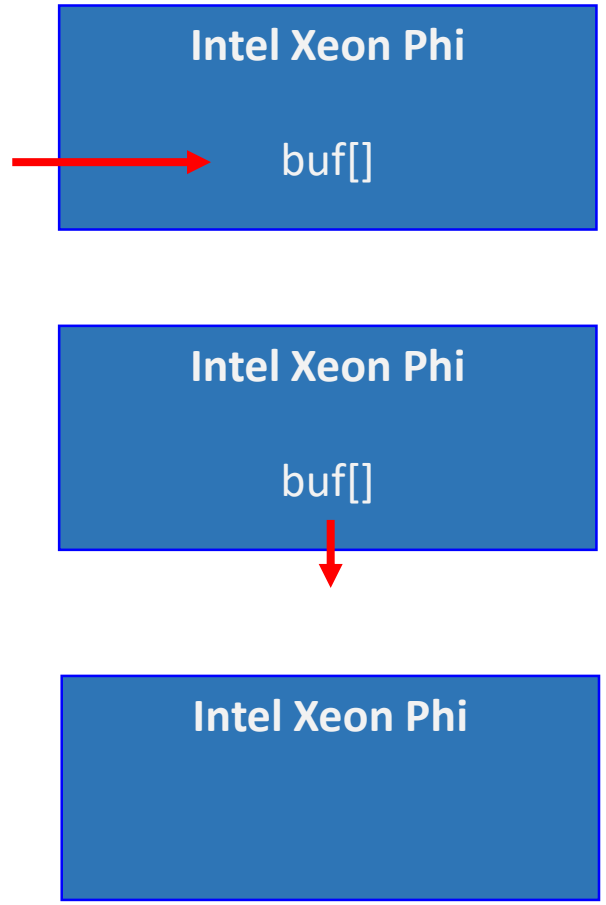
```
int main(int argc, char **argv)
{
    printf("Xeon Phi bwtest: nreps = %d\n", NREPS);
    printf("size        alignment  talloc  tsend  trecv\n");

    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

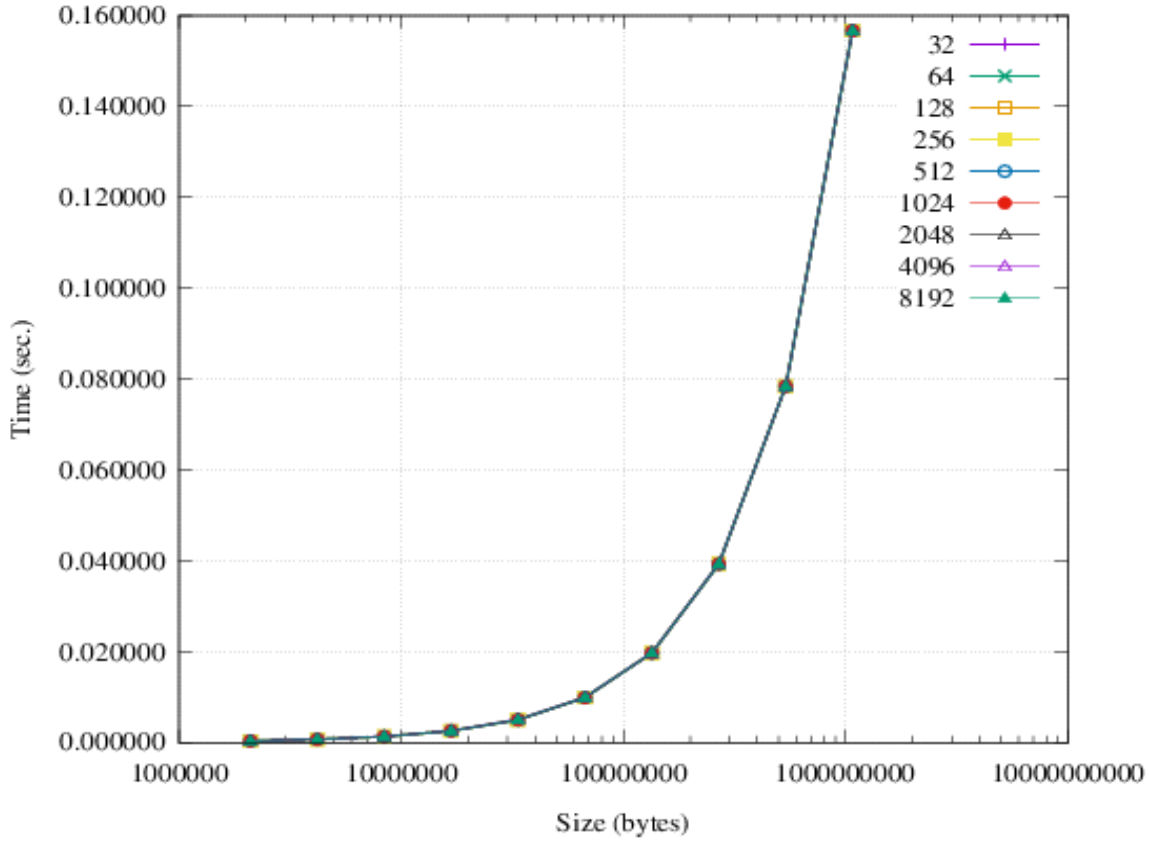
```
void testbw(int size, int alignment)
{
    uint8_t *buf = _mm_malloc(size, alignment);
    if (!buf) {
        fprintf(stderr, "Can't allocate memory\n");
        exit(EXIT_FAILURE);
    }

    // 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



Native Mode

- **Запуск программы в Native Mode**
  1. Программа компилируется на хост-машине кросс-компилятором (icc): sequential code, OpenMP, Intel Cilk Plus, Intel TBB, Intel MPI
  2. Исполняемый файл и все библиотеки копируются в Intel Xeon Phi (scp, NFS)
  3. Программа запускается на Intel Xeon Phi (scp, ssh)

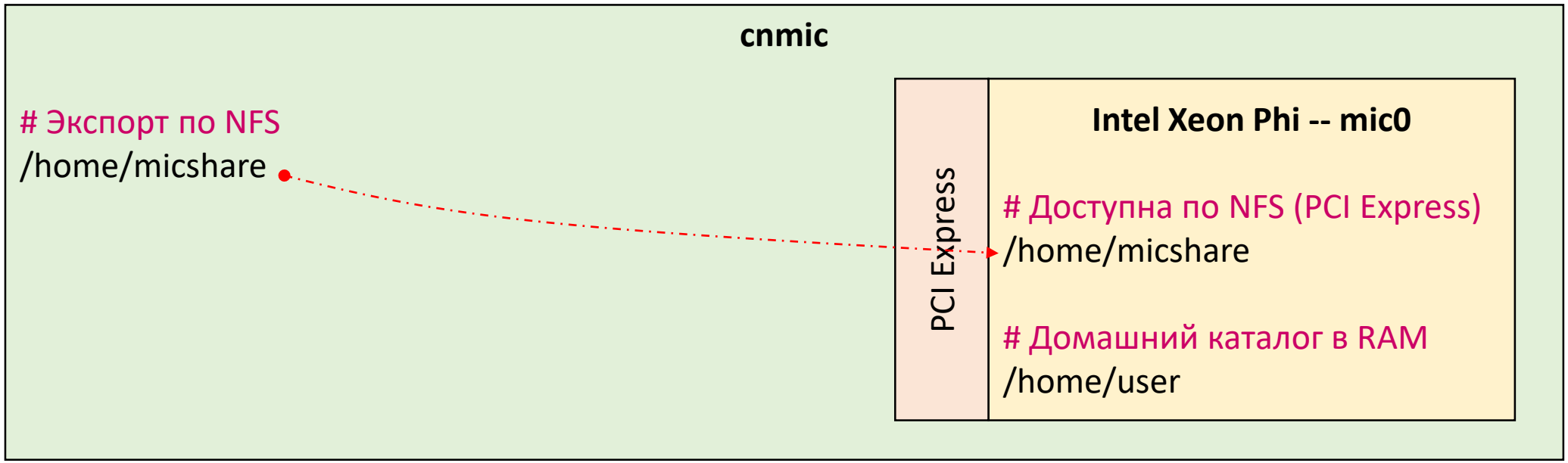
- **Ограничения**
  - ❑ Объем памяти Intel Xeon Phi (Phi 3120A GDDR5 6 GiB)
  - ❑ Отсутствует энергонезависимое хранилище – HDD/SSD
    (по NFS смонтирован каталог /home/micshare – виртуальная сеть через PCI Express)

Умножение матриц SGEMM (OpenMP)

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

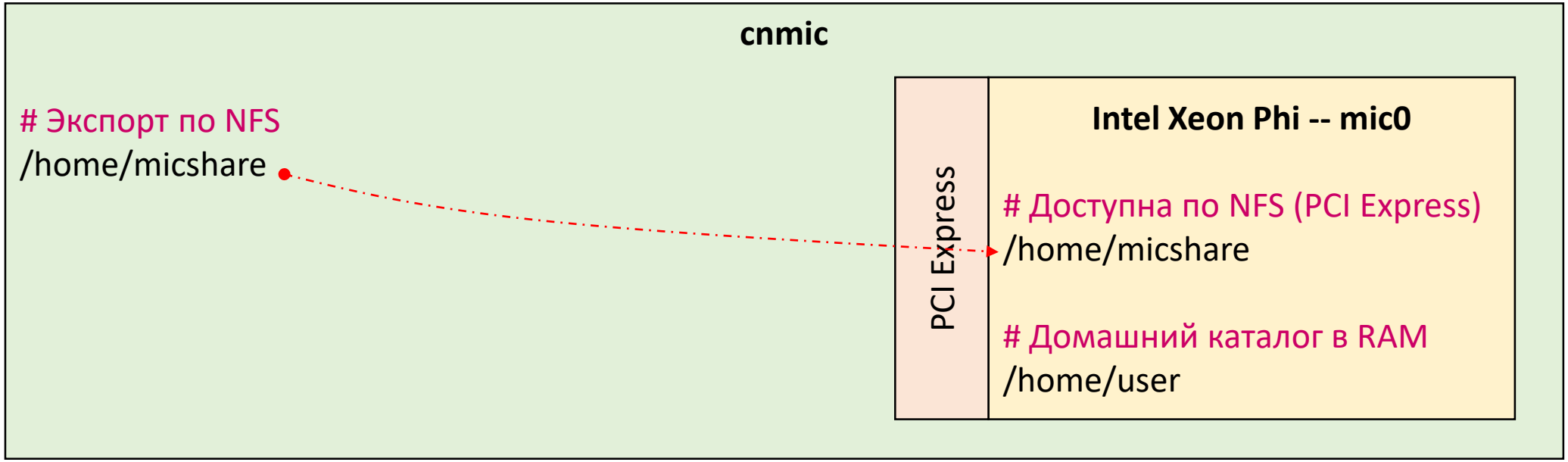
```
# Сборка с ключом -mmic
$ icc -mmic -Wall -g -std=c99 -O3 -fopenmp -c sgemm.c -o sgemm.o
$ icc -o sgemm sgemm.o -mmic -lm –fopenmp
```

Копирование исполняемого файла в Intel Xeon Phi




```
# Копирование в домашний каталог
$ scp ./sgemm mic0:
Password:
sgemm                                                    100%   50KB  50.1KB/s   00:00

# Запуск на Xeon Phi
$ ssh mic0 ./sgemm
Password:
./sgemm: error while loading shared libraries: libiomp5.so: cannot open shared object file: No such file or directory
```

Копирование исполняемого файла в Intel Xeon Phi




```
# Копирование и запуск утилитой micnativeloadex
$ cat ./launch.sh
#!/bin/sh

export SINK_LD_LIBRARY_PATH=/home/micshare/libmic
micnativeloadex ./sgemm -d 0 -e "OMP_NUM_THREADS=672 KMP_AFFINITY=explicit,granularity=fine,proclist=[1-224:1]"

$ ./launch.sh
SGEMM N = 1000, M = 1000, Q = 1000, Xeon Phi memory usage 11 MiB (0.19 %)
Phi OMP Native 672 (3 runs): perf 64.11 GFLOPS; time: tavg 0.031198, tmin 0.029922, tmax 0.032041, twarmup 1.061702
```

Intel Xeon Phi: AVX-512

```
void distance(float *x, float *y, float *z, float *d, int n)
{
    for (int i = 0; i < n; i++) {
        d[i] = sqrtf(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);
    }
}

void distance_vec_mic(float *x, float *y, float *z, float *d, int n)
{
    __m512 *xx = (__m512 *)x;
    __m512 *yy = (__m512 *)y;
    __m512 *zz = (__m512 *)z;
    __m512 *dd = (__m512 *)d;

    int k = n / 16;
    for (int i = 0; i < k; i++) {
        __m512 t1 = _mm512_mul_ps(xx[i], xx[i]);
        __m512 t2 = _mm512_mul_ps(yy[i], yy[i]);
        __m512 t3 = _mm512_mul_ps(zz[i], zz[i]);
        t1 = _mm512_add_ps(t1, t2);
        t1 = _mm512_add_ps(t1, t3);
        dd[i] = _mm512_sqrt_ps(t1);
    }
    for (int i = k * 16; i < n; i++) {
        d[i] = sqrtf(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);
    }
}
```

OpenMP + SIMD

```
void distance(float *x, float *y, float *z, float *d, int n)
{
    // SIMD only
    #pragma omp simd
    for (int i = 0; i < n; i++) {
        d[i] = sqrtf(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);
    }
}

void distance(float *x, float *y, float *z, float *d, int n)
{
    // Threading + SIMD
    #pragma omp parallel for simd
    for (int i = 0; i < n; i++) {
        d[i] = sqrtf(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);
    }
}
```

http://primeurmagazine.com/repository/PrimeurMagazine-AE-PR-12-14-32.pdf

Native Mode

- **Запуск программы в Native Mode**
  1. Программа компилируется на хост-машине кросс-компилятором (icc): sequential code, OpenMP, Intel Cilk Plus, Intel TBB, Intel MPI
  2. Исполняемый файл и все библиотеки копируются в Intel Xeon Phi (scp, NFS)
  3. Программа запускается на Intel Xeon Phi (scp, ssh)

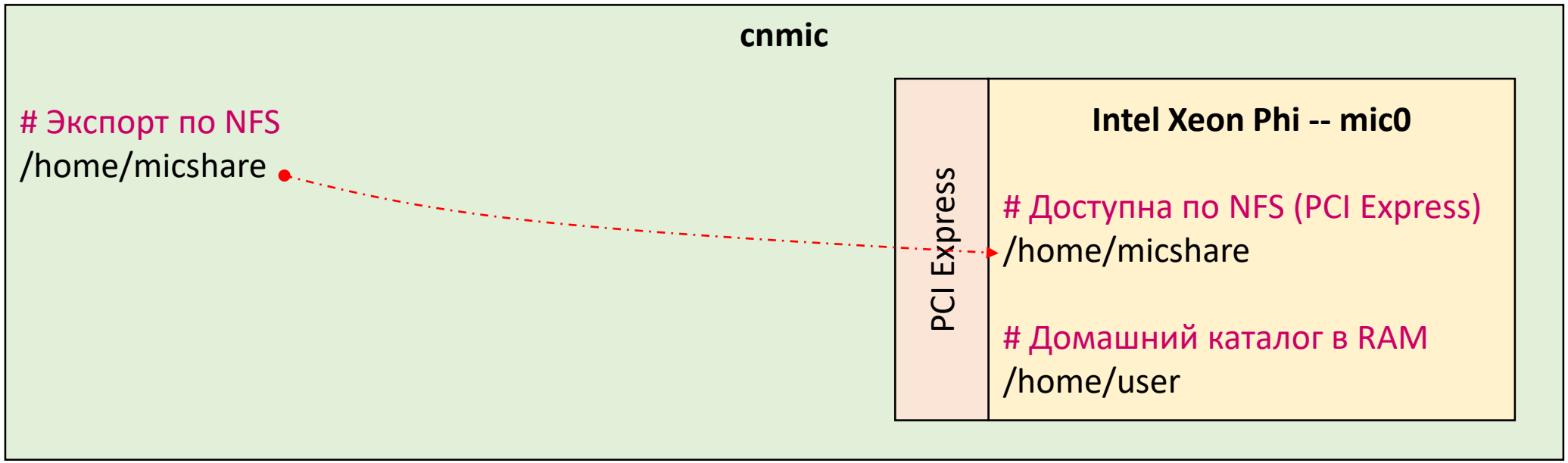
- **Ограничения**
  - ❑ Объем памяти Intel Xeon Phi (Phi 3120A GDDR5 6 GiB)
  - ❑ Отсутствует энергонезависимое хранилище – HDD/SSD
    (по NFS смонтирован каталог /home/micshare – виртуальная сеть через PCI Express)

Умножение матриц SGEMM (OpenMP)

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

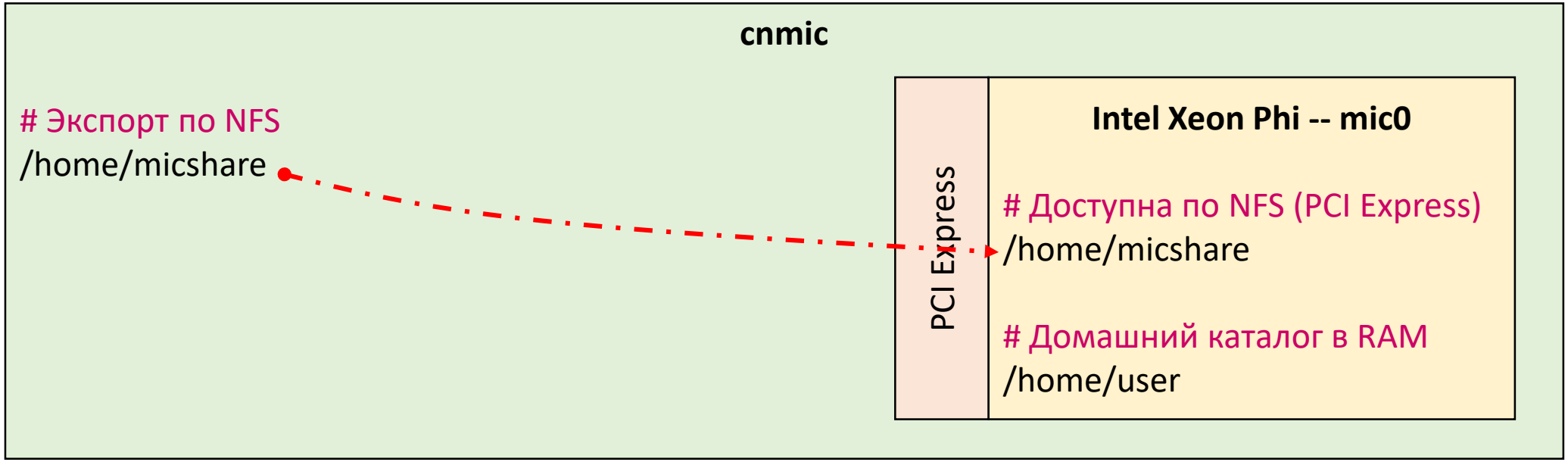
```
# Сборка с ключом -mmic
$ icc -mmic -Wall -g -std=c99 -O3 -fopenmp -c sgemm.c -o sgemm.o
$ icc -o sgemm sgemm.o -mmic -lm –fopenmp
```

Копирование исполняемого файла в Intel Xeon Phi




```
# Копирование в домашний каталог
$ scp ./sgemm mic0:
Password:
sgemm                                                    100%   50KB  50.1KB/s   00:00

# Запуск на Xeon Phi
$ ssh mic0 ./sgemm
Password:
./sgemm: error while loading shared libraries: libiomp5.so: cannot open shared object file: No such file or directory
```

Копирование исполняемого файла в Intel Xeon Phi




```
# Копирование и запуск утилитой micnativeloadex
$ cat ./launch.sh
#!/bin/sh

export SINK_LD_LIBRARY_PATH=/home/micshare/libmic
micnativeloadex ./sgemm -d 0 -e "OMP_NUM_THREADS=672 KMP_AFFINITY=explicit,granularity=fine,proclist=[1-224:1]"

$ ./launch.sh
SGEMM N = 1000, M = 1000, Q = 1000, Xeon Phi memory usage 11 MiB (0.19 %)
Phi OMP Native 672 (3 runs): perf 64.11 GFLOPS; time: tavg 0.031198, tmin 0.029922, tmax 0.032041, twarmup 1.061702
```

Intel Xeon Phi: AVX-512

```
void distance(float *x, float *y, float *z, float *d, int n)
{
    for (int i = 0; i < n; i++) {
        d[i] = sqrtf(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);
    }
}

void distance_vec_mic(float *x, float *y, float *z, float *d, int n)
{
    __m512 *xx = (__m512 *)x;
    __m512 *yy = (__m512 *)y;
    __m512 *zz = (__m512 *)z;
    __m512 *dd = (__m512 *)d;

    int k = n / 16;
    for (int i = 0; i < k; i++) {
        __m512 t1 = _mm512_mul_ps(xx[i], xx[i]);
        __m512 t2 = _mm512_mul_ps(yy[i], yy[i]);
        __m512 t3 = _mm512_mul_ps(zz[i], zz[i]);
        t1 = _mm512_add_ps(t1, t2);
        t1 = _mm512_add_ps(t1, t3);
        dd[i] = _mm512_sqrt_ps(t1);
    }
    for (int i = k * 16; i < n; i++) {
        d[i] = sqrtf(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);
    }
}
```

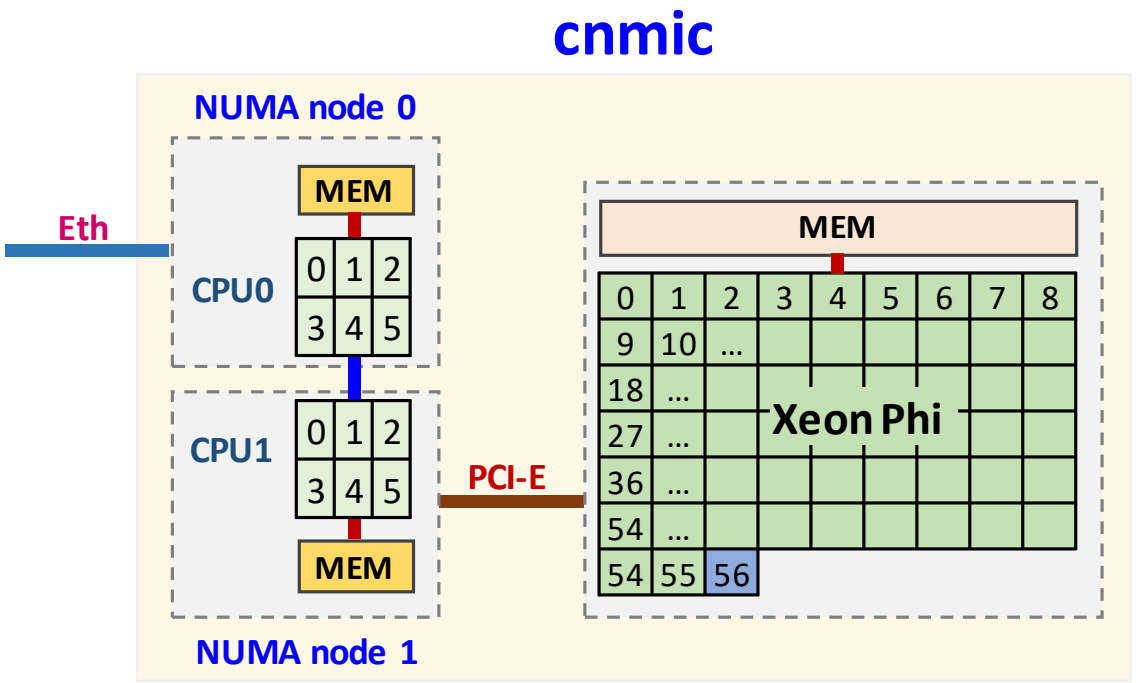
OpenMP + SIMD

```
void distance(float *x, float *y, float *z, float *d, int n)
{
    // SIMD only
    #pragma omp simd
    for (int i = 0; i < n; i++) {
        d[i] = sqrtf(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);
    }
}

void distance(float *x, float *y, float *z, float *d, int n)
{
    // Threading + SIMD
    #pragma omp parallel for simd
    for (int i = 0; i < n; i++) {
        d[i] = sqrtf(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);
    }
}
```

http://primeurmagazine.com/repository/PrimeurMagazine-AE-PR-12-14-32.pdf

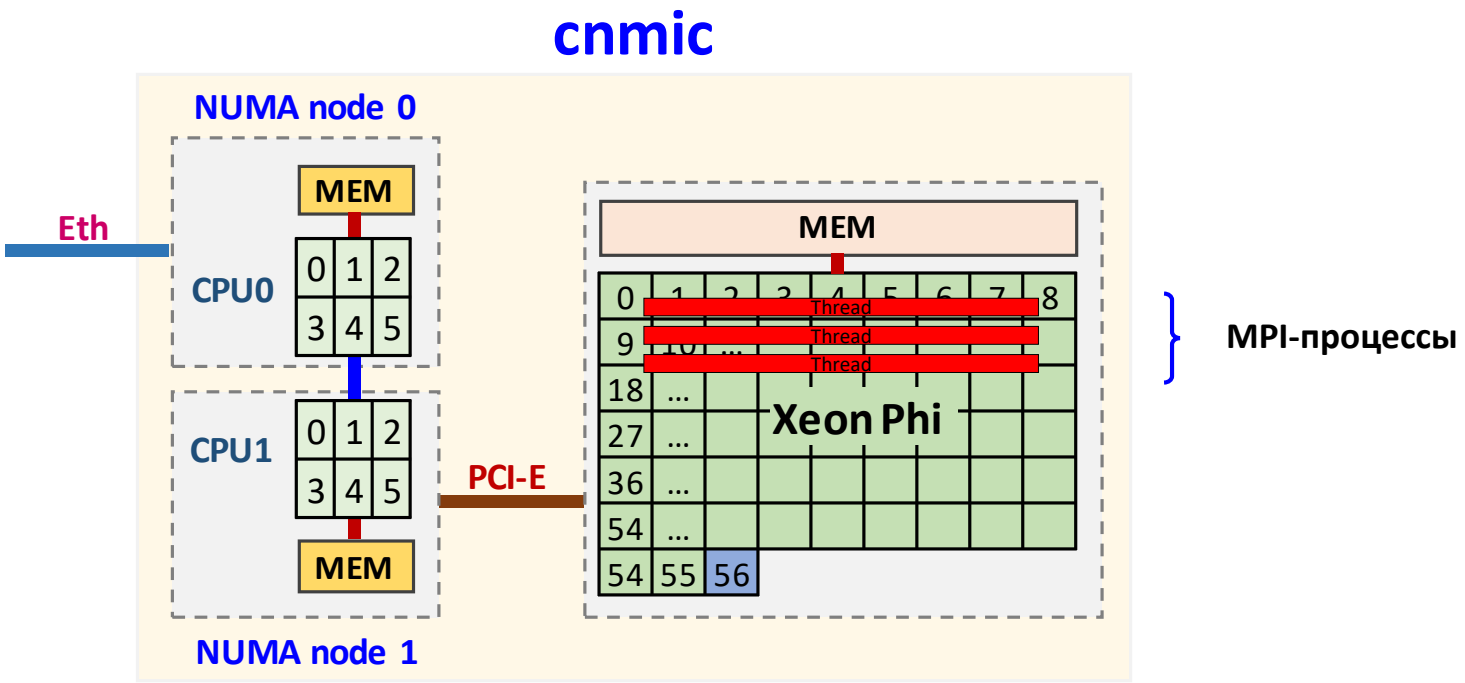
Использование MPI на Intel Xeon Phi

- **Native model** – все процессы MPI-программы выполняются на сопроцессоре
- Программа копируется на сопроцессор и запускается



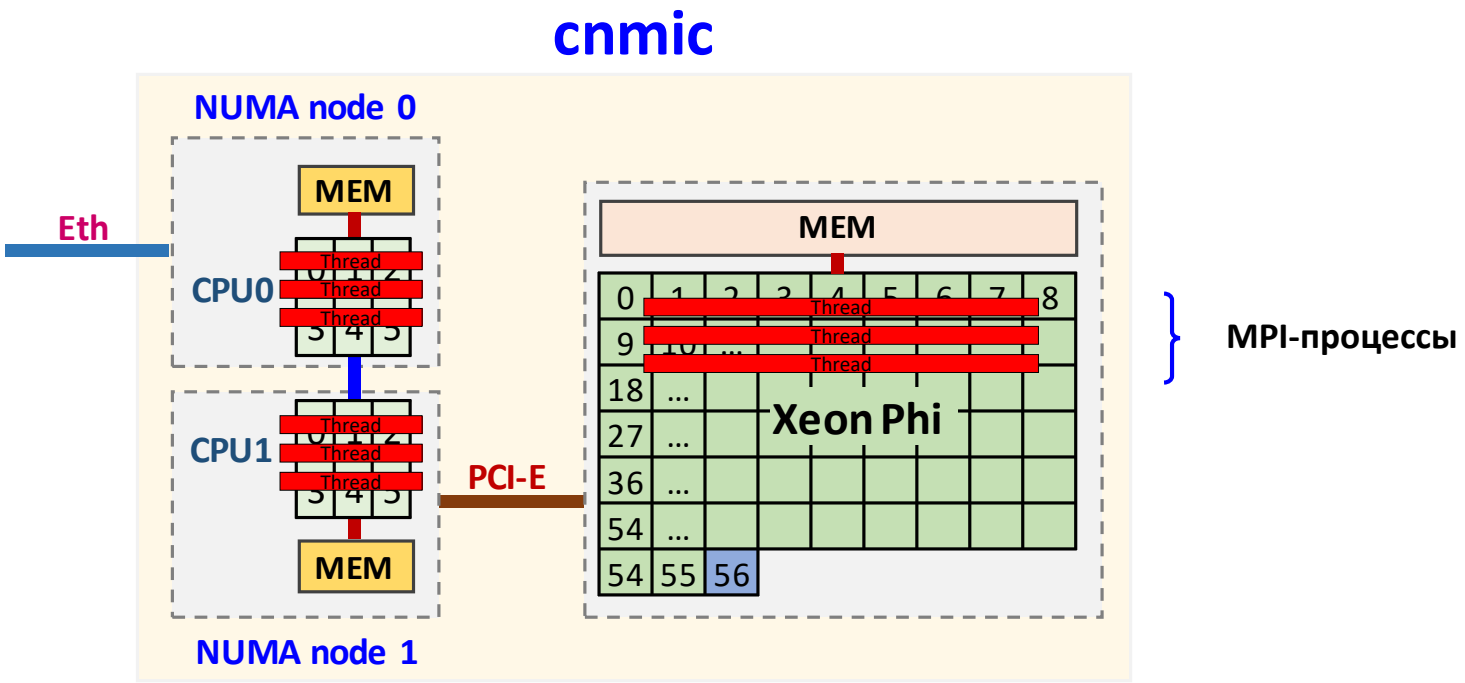
Использование MPI на Intel Xeon Phi

- **Native model** – все процессы MPI-программы выполняются на сопроцессоре
- Программа копируется на сопроцессор и запускается



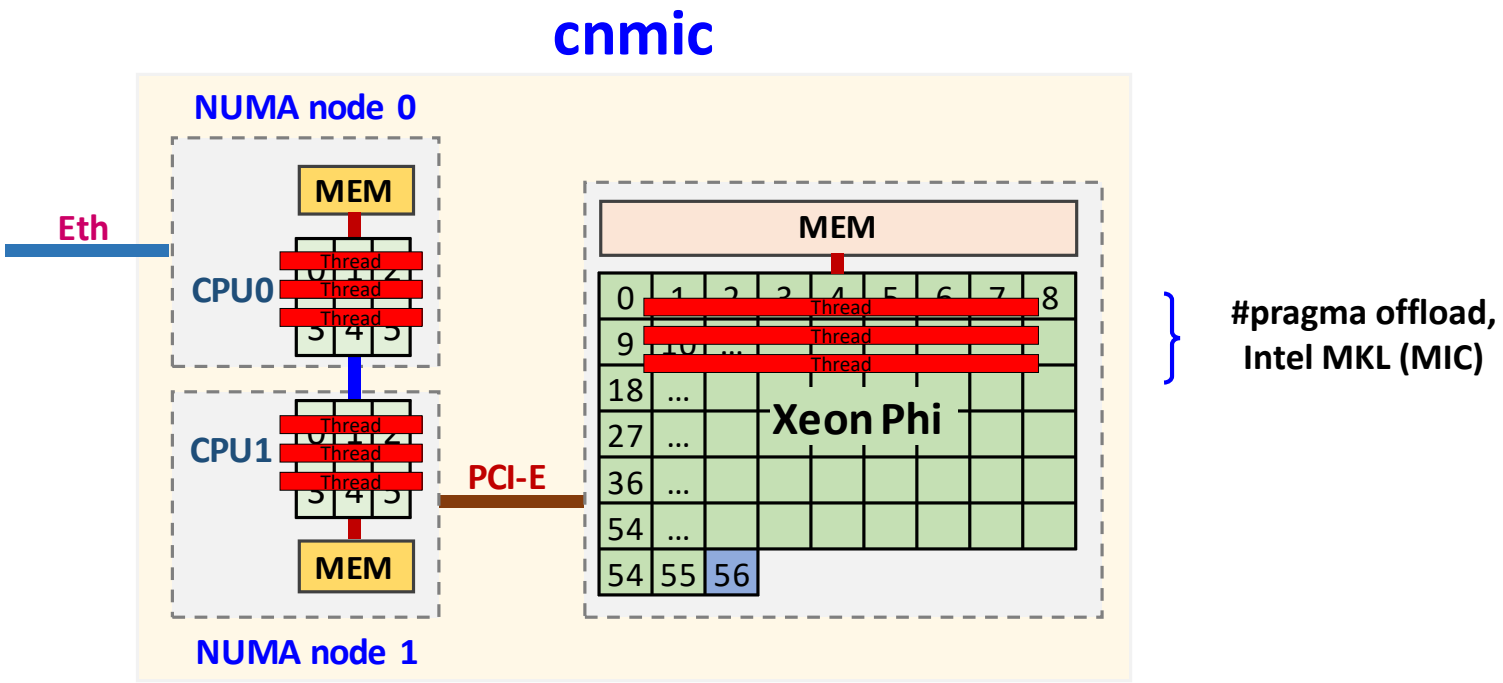
Использование MPI на Intel Xeon Phi

- **Symmetric model** – процессы программы выполняются на хост-машине и сопроцессоре



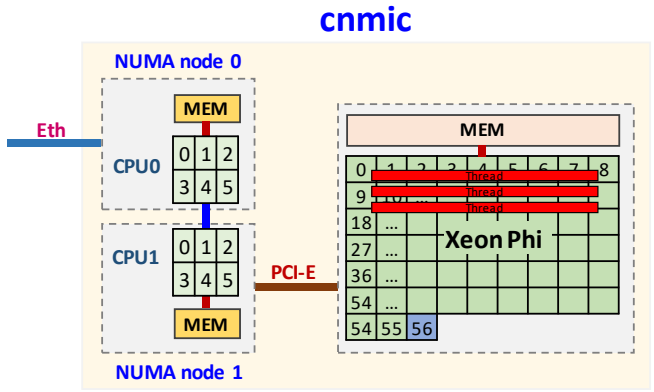
Использование MPI на Intel Xeon Phi

- **Offload model** – процессы MPI-программы выполняются на хост-машине и выгружают (offload) часть кода на сопроцессор
- #pragma offload, Intel MKL (MIC)



MPI Native mode

```c
#include <unistd.h>
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv)
{
    int rank, len;
    char procname[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Get_processor_name(procname, &len);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    printf("Rank %d on %s\n", rank, procname);

    MPI_Finalize();
    return 0;
}
```




```
# Сборка программы
$ cat ./build.sh
#!/bin/sh

mpiicc -mmic -Wall -std=c99 -O2 -o \
        hello ./hello.c
```

```
# Запуск программы
$ cat ./launch.sh
#!/bin/sh

TMP=`mktemp -p /home/micshare/tmp -u`
cp -f ./hello $TMP
export I_MPI_MIC=enable

mpirun -n 3 -f hosts $TMP

rm -f $TMP
```

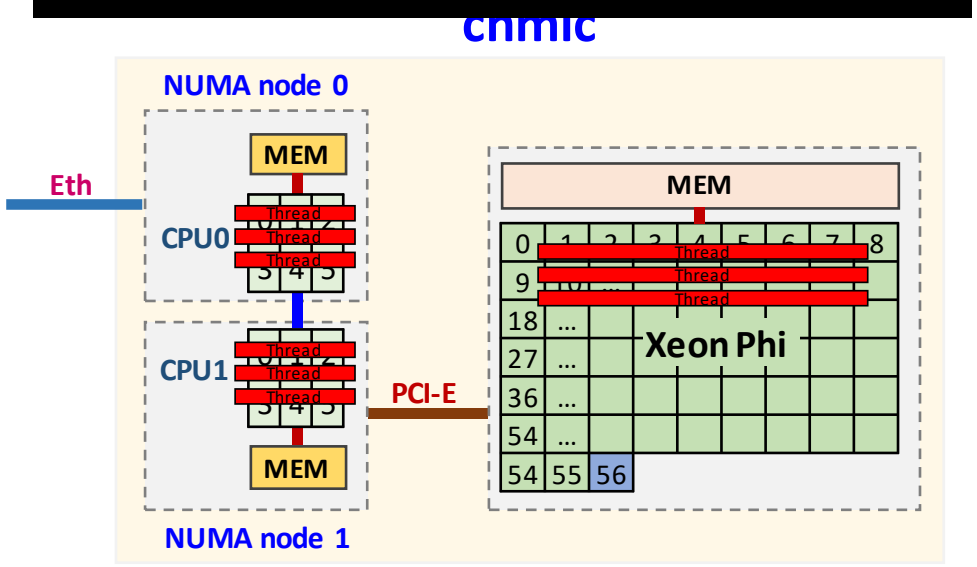
mic0

```
$ ./launch.sh
Rank 2 on cnmic-mic0
Rank 0 on cnmic-mic0
Rank 1 on cnmic-mic0
```

MPI Symmetric mode

```
#include <unistd.h>
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv)
```

```
$ ./launch.sh
Rank 0 on cnmic
Rank 1 on cnmic
Rank 2 on cnmic-mic0
Rank 4 on cnmic-mic0
Rank 3 on cnmic-mic0
```



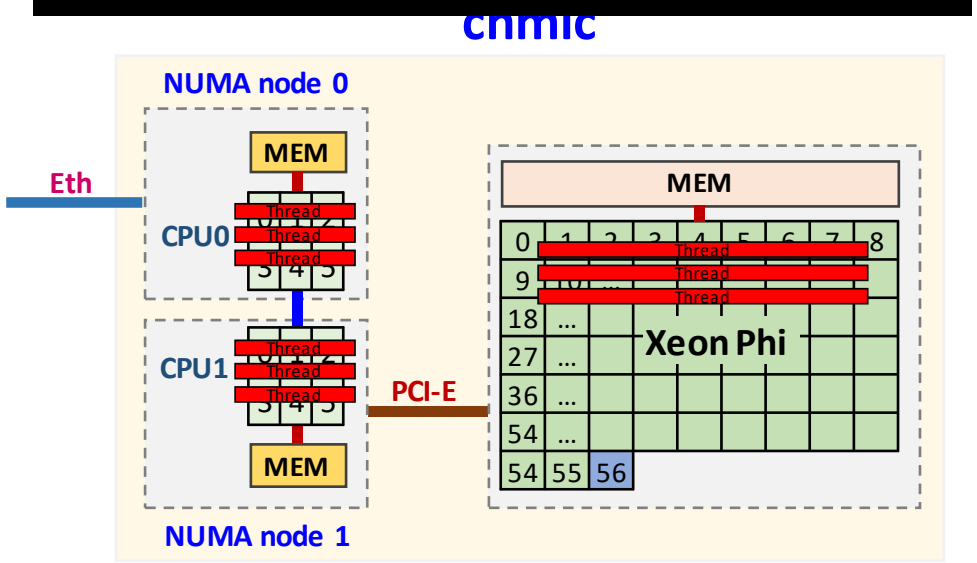

```
# Сборка программы
$ cat ./build.sh
#!/bin/sh

# Binary for Xeon Phi
mpiicc -mmic -Wall -std=c99 -O2 -o hello.mic ./hello.c

# Binary for Host
mpiicc -Wall -std=c99 -O2 -o hello ./hello.c
```

```
# Запуск программы Host + Phi – Вариант 1
$ cat ./build.sh
#!/bin/sh
# Copy program to the NFS folder
TMP=`mktemp -p /home/micshare/tmp -u`
cp -f ./hello.mic $TMP

# Launch the program (see hosts file)
export I_MPI_MIC=enable
export I_MPI_FABRICS=shm:tcpmpi
run -n 2 -host cnmic ./hello : -n 3 -host mic0 $TMP

rm -f $TMP
```

MPI Symmetric mode

```
#include <unistd.h>
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv)
```

```
$ ./launch.sh
Rank 0 on cnmic
Rank 1 on cnmic
Rank 2 on cnmic-mic0
Rank 4 on cnmic-mic0
Rank 3 on cnmic-mic0
```




```
# Сборка программы
$ cat ./build.sh
#!/bin/sh

# Binary for Xeon Phi
mpiicc -mmic -Wall -std=c99 -O2 -o hello.mic ./hello.c

# Binary for Host
mpiicc -Wall -std=c99 -O2 -o hello ./hello.c
```

```
# Запуск программы Host + Phi – Вариант 2
$ cat ./build.sh
# Copy program to the NFS folder
TMP=`mktemp -p /home/micshare/tmp -u`
cp -f ./hello.mic "$TMP.mic"
cp -f ./hello $TMP
# Launch the program (see hosts file)
export I_MPI_MIC=enable
export I_MPI_MIC_POSTFIX=.mic
export I_MPI_FABRICS=shm:tcp
mpirun -machinefile hosts $TMP

rm -f $TMP "$TMP.mic"
```

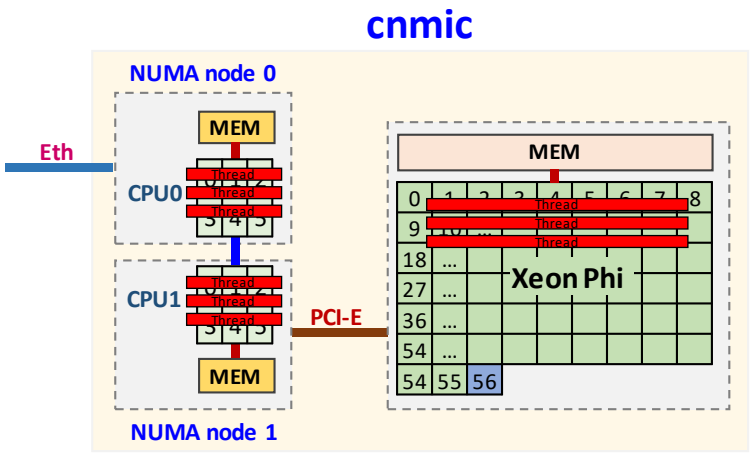
```
cnmic:2
mic0:3
```

MPI Offload mode

```c
int main(int argc, char **argv)
{
    int rank, len;
    char procname[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Get_processor_name(procname, &len);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    printf("Rank %d on %s\n", rank, procname);

    #pragma offload target(mic) in(rank)
    {
        printf("Hello, from Xeon Phi (offloaded from proc %d)\n", rank);
    }

    MPI_Finalize();
    return 0;
}
```

```
# Сборка программы
$ cat ./build.sh
#!/bin/sh

mpiicc -Wall -std=c99 -O2 -o hello ./hello.c
```

```
# Запуск программы
$ cat ./launch.sh
#!/bin/sh

mpirun -n 4 -host cnmic ./hello ./hello
```




```
$ ./launch.sh
Rank 0 on cnmic
Rank 1 on cnmic
Rank 2 on cnmic
Rank 3 on cnmic
Hello, from Xeon Phi (offloaded from proc 3)
Hello, from Xeon Phi (offloaded from proc 0)
Hello, from Xeon Phi (offloaded from proc 2)
Hello, from Xeon Phi (offloaded from proc 1)
```

Конфигурация гибридного NUMA-узла cnmic

```
$ hwloc-ls
Machine (64GB)
  NUMANode L#0 (P#0 32GB)
    Socket L#0 + L3 L#0 (15MB)
      L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
    HostBridge L#0
      PCIBridge
        PCI 10de:0f02
          GPU L#0 "card0"
          GPU L#1 "renderD128"
          GPU L#2 "controlD64"
      PCI 8086:8d62
      PCIBridge
        PCI 8086:1533
          Net L#3 "enp6s0"
      PCIBridge
        PCI 8086:1533
          Net L#4 "enp7s0"
      PCIBridge
        PCI 1b21:0612
      PCI 8086:8d02
        Block L#5 "sda"
  NUMANode L#1 (P#1 32GB)
    Socket L#1 + L3 L#1 (15MB)
      L2 L#6 (256KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6
    HostBridge L#5
      PCIBridge
        PCI 8086:225d
          CoProc L#6 "mic0"
```