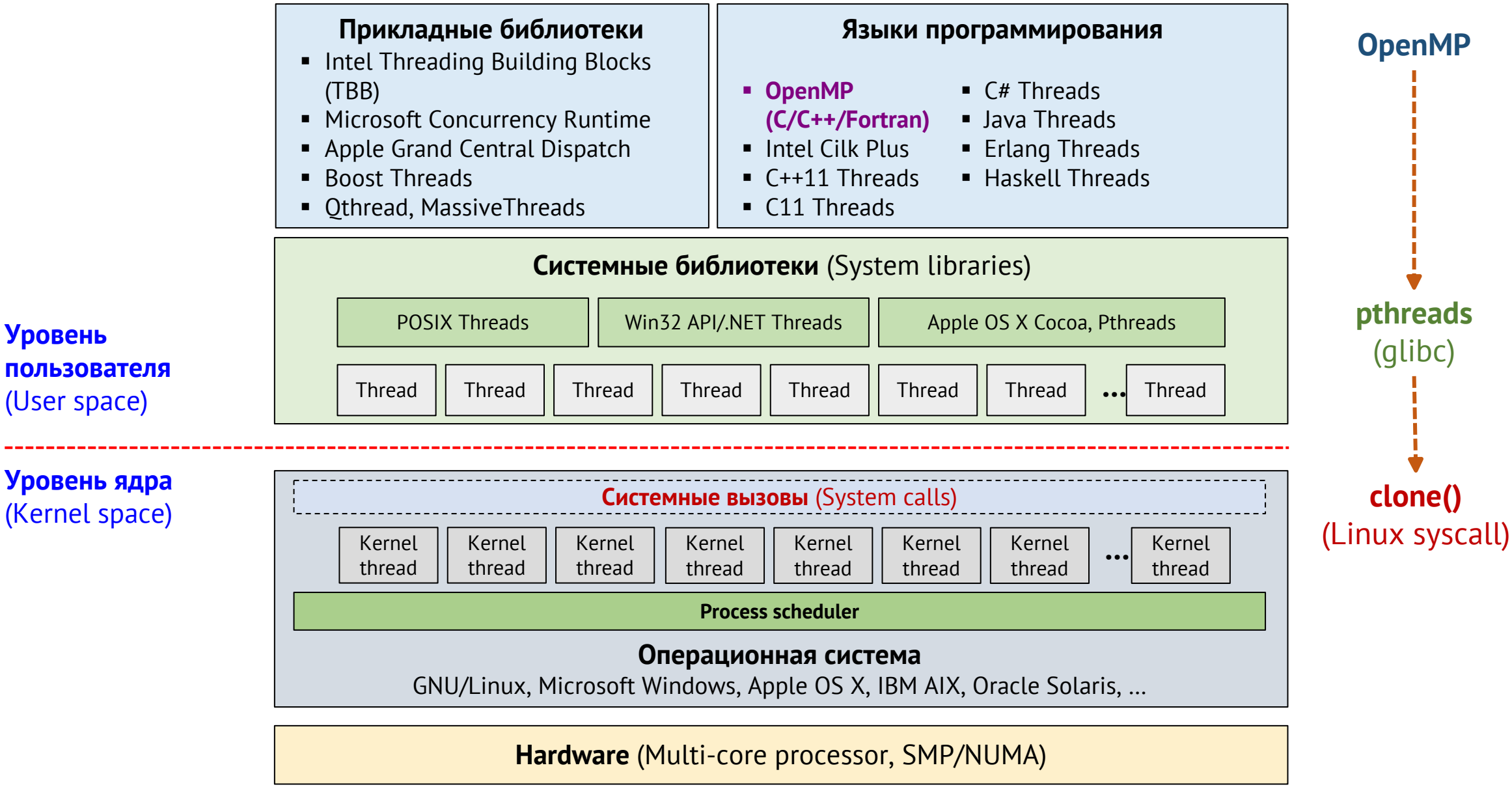
Лекция 5
Стандарт OpenMP: умножение матрицы на вектор

Курносов Михаил Георгиевич

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

Курс «Параллельные вычислительные технологии»
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Средства многопоточного программирования



Стандарт OpenMP

- **OpenMP (Open Multi-Processing)** – стандарт, определяющий набор директив компилятора, библиотечных процедур и переменных среды окружения для создания многопоточных программ

- Разрабатывается в рамках OpenMP Architecture Review Board с 1997 года
  - OpenMP 2.5 (2005), OpenMP 3.0 (2008), OpenMP 4.5 (2015), OpenMP 5.0 (2018), **OpenMP 5.2 (2021)**
  - http://www.openmp.org
  - https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf

- Требуется поддержка со стороны компилятора

Стандарт OpenMP

https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf

OpenMP

OpenMP
Application Programming
Interface

Version 5.2 November 2021

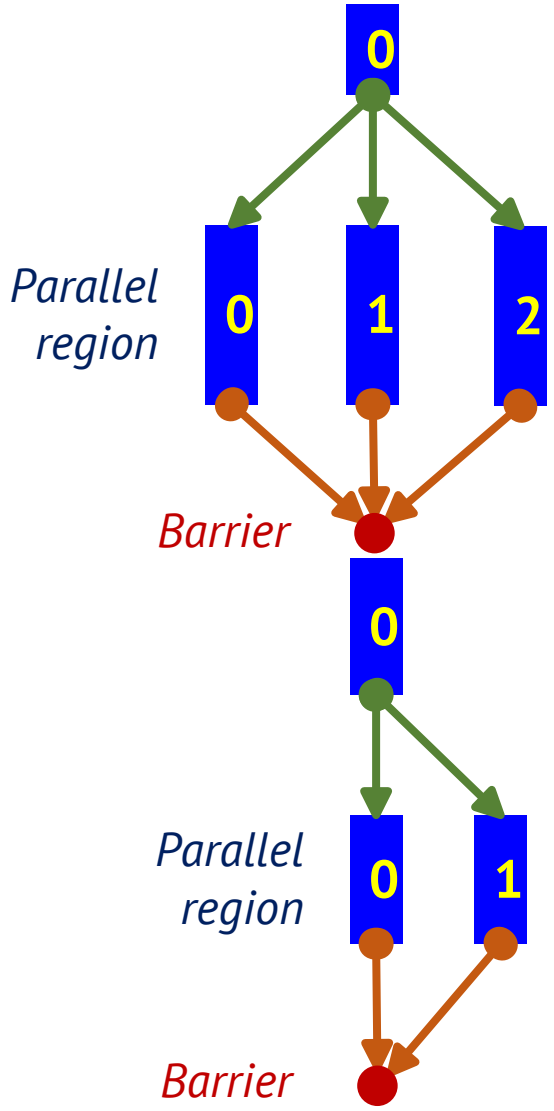
Contents

Поддержка компиляторами

Compiler	Information
GNU GCC https://gcc.gnu.org/wiki/openmp	Option: –fopenmp gcc 4.7 – OpenMP 3.1 gcc 4.9 – OpenMP 4.0 gcc 5.x – Offloading gcc 11 — OpenMP 4.5 >
Clang (LLVM) http://openmp.llvm.org/	Option: –fopenmp Clang 3.8.0 – OpenMP 3.1 Clang + Intel OpenMP RTL (http://clang-omp.github.io/)
Intel C/C++, Fortran https://software.intel.com/en-us/c-compilers/	OpenMP 4.x Option: –Qopenmp, –openmp
Oracle Solaris Studio C/C++/Fortran	OpenMP 4.0 Option: –xopenmp
Microsoft Visual Studio C++	Option: /openmp OpenMP 2.0 only
http://www.openmp.org/resources/openmp-compilers/	

Модель выполнения OpenMP-программы

- **Динамическое управление потоками в модели Fork-Join**:
  - ✓ **Fork** – порождение нового потока
  - ✓ **Join** – ожидание завершения потока (объединение потоков управления)
- OpenMP-программа – совокупность последовательных участков кода (serial code) и параллельных регионов (parallel region)
- Каждый поток имеет логический номер: 0, 1, 2, …
- Главный поток (master) имеет номер 0
- Параллельные регионы могут быть вложенными



OpenMP: Hello, World

```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv)
{
    #pragma omp parallel    /* <-- Fork */
    {
        printf("Hello, multithreaded world: thread %d of %d\n",
               omp_get_thread_num(), omp_get_num_threads());
    }                       /* <-- Barrier & join */

   return 0;
}
```

Компиляция и запуск OpenMP-программы

```
$ gcc -fopenmp -o hello ./hello.c

$ ./hello
Hello, multithreaded world: thread 0 of 4
Hello, multithreaded world: thread 1 of 4
Hello, multithreaded world: thread 3 of 4
Hello, multithreaded world: thread 2 of 4
```

- По умолчанию количество потоков в параллельном регионе равно числу логических процессоров в системе
- Порядок выполнения потоков заранее неизвестен – определяется планировщиком операционной системы

Указание числа потоков в параллельных регионах

```
$ export OMP_NUM_THREADS=8

$ ./hello
Hello, multithreaded world: thread 1 of 8
Hello, multithreaded world: thread 2 of 8
Hello, multithreaded world: thread 3 of 8
Hello, multithreaded world: thread 0 of 8
Hello, multithreaded world: thread 4 of 8
Hello, multithreaded world: thread 5 of 8
Hello, multithreaded world: thread 6 of 8
Hello, multithreaded world: thread 7 of 8
```

Задание числа потоков в параллельном регионе

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv)
{
    #pragma omp parallel num_threads(6)
    {
        printf("Hello, multithreaded world: thread %d of %d\n",
               omp_get_thread_num(), omp_get_num_threads());
    }

    return 0;
}
```

Задание числа потоков в параллельном регионе

```
$ export OMP_NUM_THREADS=8

$ ./hello
Hello, multithreaded world: thread 2 of 6
Hello, multithreaded world: thread 3 of 6
Hello, multithreaded world: thread 1 of 6
Hello, multithreaded world: thread 0 of 6
Hello, multithreaded world: thread 4 of 6
Hello, multithreaded world: thread 5 of 6
```

- Директива num_threads имеет приоритет над значением переменной среды окружения OMP_NUM_THREADS

Список потоков процесса

```c
#include <stdio.h>
#include <omp.h>
#include <time.h>

int main(int argc, char **argv)
{
    #pragma omp parallel num_threads(6)
    {
        printf("Hello, multithreaded world: thread %d of %d\n",
               omp_get_thread_num(), omp_get_num_threads());

        /* Sleep for 30 seconds */
        nanosleep(&(struct timespec){.tv_sec = 30}, NULL);
    }
    return 0;
}
```

Список потоков процесса

```
$ ./hello &
$ ps -eLo pid,tid,psr,args | grep hello
 6157  6157   0 ./hello
 6157  6158   1 ./hello
 6157  6159   0 ./hello
 6157  6160   1 ./hello
 6157  6161   0 ./hello
 6157  6162   1 ./hello
 6165  6165   2 grep hello
```

- **Номер процесса (PID)**
- **Номер потока (TID)**
- **Логический процессор (PSR)**
- **Название исполняемого файла**

- **Информация о логических процессорах системы:**
  - /proc/cpuinfo
  - /sys/devices/system/cpu

#pragma omp parallel

10 Parallelism Generation and Control

This chapter defines constructs for generating and controlling parallelism.

10.1 `parallel` Construct

Name: `parallel` **Category:** executable	**Association:** block **Properties:** parallelism-generating, cancellable, thread-limiting, context-matching

Clauses

`allocate`, `copyin`, `default`, `firstprivate`, `if`, `num_threads`, `private`, `proc_bind`, `reduction`, `shared`

Binding

The binding thread set for a `parallel` region is the encountering thread. The encountering thread becomes the primary thread of the new team.

Semantics

When a thread encounters a `parallel` construct, a team of threads is created to execute the `parallel` region (see Section 10.1.1 for more information about how the number of threads in the team is determined, including the evaluation of the `if` and `num_threads` clauses). The thread that encountered the `parallel` construct becomes the primary thread of the new team, with a thread number of zero for the duration of the new `parallel` region. All threads in the new team, including the primary thread, execute the region. Once the team is created, the number of threads in the team remains constant for the duration of that `parallel` region.

Within a `parallel` region, thread numbers uniquely identify each thread. Thread numbers are consecutive whole numbers ranging from zero for the primary thread up to one less than the number of threads in the team. A thread may obtain its own thread number by a call to the `omp_get_thread_num` library routine.

A set of implicit tasks, equal in number to the number of threads in the team, is generated by the encountering thread. The structured block of the `parallel` construct determines the code that will be executed in each implicit task. Each task is assigned to a different thread in the team and becomes tied. The task region of the task that the encountering thread is executing is suspended and each thread in the team executes its implicit task. Each thread can execute a path of statements that is different from that of the other threads.

- team of threads
- primary thread (thread 0)
- implicit task
- tied task

```
int main(int argc, char **argv)
{
    // Создается группа из 4 потоков
    // 4 задачи – по одной на каждый поток
    #pragma omp parallel num_threads(4)
    {
        /* Code */
    }
    return 0;
}
```

Умножение матрицы на вектор

DGEMV – BLAS Level 2

(BLAS – Basic Linear Algebra Subroutines)

Умножение матрицы на вектор (DGEMV)

- Требуется вычислить произведение прямоугольной матрицы ***A*** размера $m \times n$ на вектор-столбец ***B*** размера $m \times 1$ (BLAS Level 2, DGEMV)

$$\boldsymbol{C_{m\times 1} = A_{m\times n}\cdot B_{n\times 1}}$$

$$C = \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_m \end{pmatrix} \qquad A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \qquad B = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix}$$

$$c_i = \sum_{j=1}^{n} a_{ij}\cdot b_j, \;\; i = 1,2,\dots,m.$$

DGEMV: последовательная версия

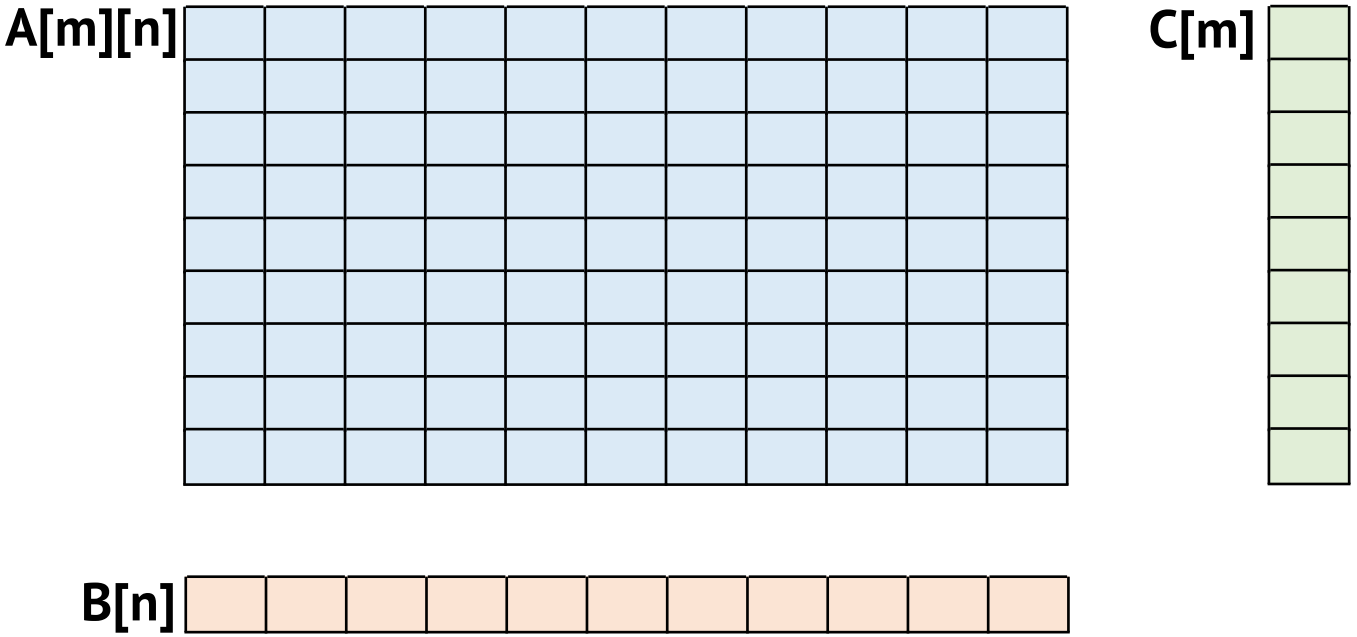
```
/*
 * matrix_vector_product: Compute matrix-vector product c[m] = a[m][n] * b[n]
 */
void matrix_vector_product(double *a, double *b, double *c, int m, int n)
{
    for (int i = 0; i < m; i++) {
        c[i] = 0.0;
        for (int j = 0; j < n; j++)
            c[i] += a[i * n + j] * b[j];
    }
}
```

$$c_i = \sum_{j=1}^{n} a_{ij} \cdot b_j, \quad i = 1,2,\ldots,m.$$

DGEMV: последовательная версия

```
void run_serial()
{
    double *a, *b, *c;

    a = xmalloc(sizeof(*a) * m * n);
    b = xmalloc(sizeof(*b) * n);
    c = xmalloc(sizeof(*c) * m);

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++)
            a[i * n + j] = i + j;
    }
    for (int j = 0; j < n; j++)
        b[j] = j;

    double t = wtime();
    matrix_vector_product(a, b, c, m, n);
    t = wtime() - t;

    printf("Elapsed time (serial): %.6f sec.\n", t);
    free(a);
    free(b);
    free(c);
}
```

DGEMV: параллельная версия

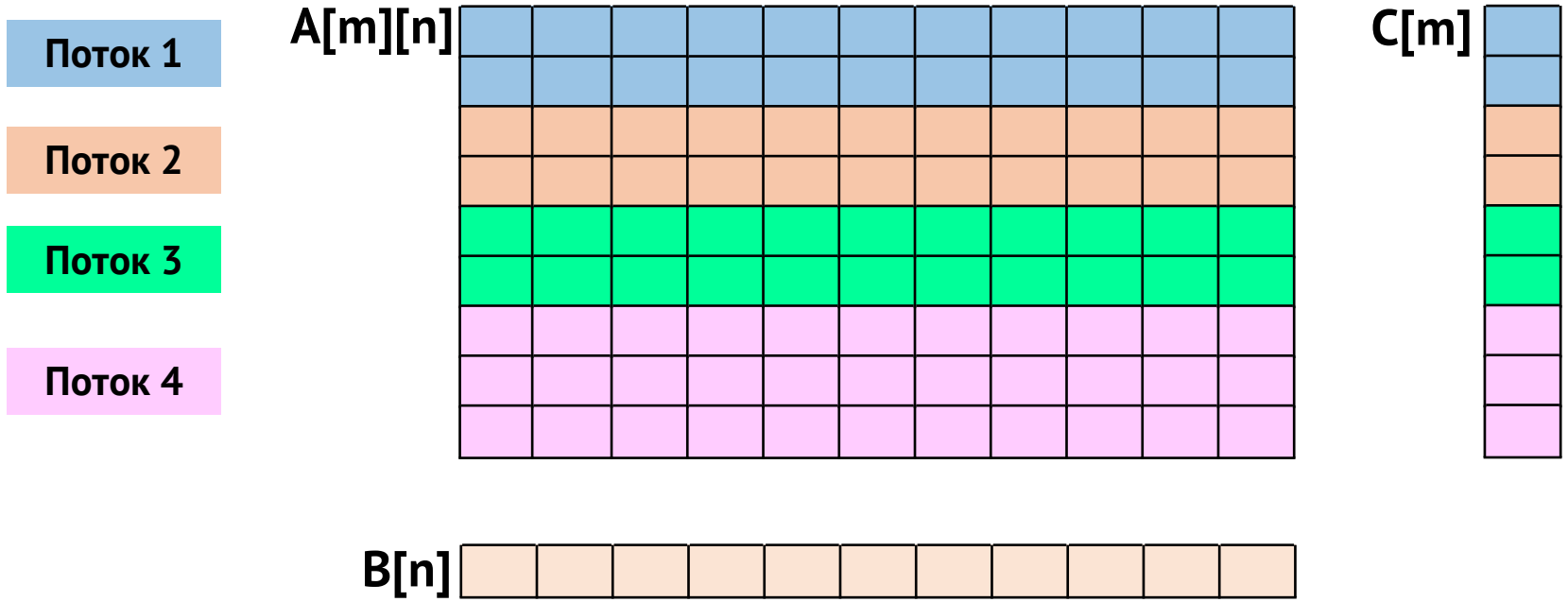



```
for (int i = 0; i < m; i++) {
    c[i] = 0.0;
    for (int j = 0; j < n; j++)
        c[i] += a[i * n + j] * b[j];
}
```

Требования к параллельному алгоритму

- Максимальная загрузка потоков вычислениями
- Минимум совместно используемых ячеек памяти – независимые области данных

DGEMV: параллельная версия



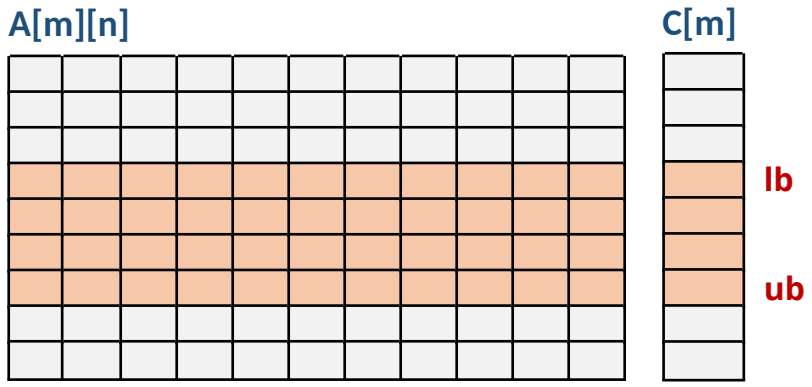

```
for (int i = 0; i < m; i++) {
    c[i] = 0.0;
    for (int j = 0; j < n; j++)
        c[i] += a[i * n + j] * b[j];
}
```

Распараллеливание внешнего цикла

- Каждый поток вычисляет *m/p* последовательных расположенных элементов вектора *C*[*i*]
- Каждый поток читает элементы из горизонтальной полосы матрицы *A*[*i*,*j*], а также все элементы вектора *B*[*j*]

DGEMV: параллельная версия

```
/* matrix_vector_product_omp: Compute matrix-vector product c[m] = a[m][n] * b[n] */
void matrix_vector_product_omp(double *a, double *b, double *c, int m, int n)
{
    #pragma omp parallel
    {
        int nthreads = omp_get_num_threads();
        int threadid = omp_get_thread_num();
        int items_per_thread = m / nthreads;
        int lb = threadid * items_per_thread;
        int ub = (threadid == nthreads - 1) ? (m - 1) : (lb + items_per_thread - 1);

        for (int i = lb; i <= ub; i++) {
            c[i] = 0.0;
            for (int j = 0; j < n; j++)
                c[i] += a[i * n + j] * b[j];
        }
    }
}
```



- Каждый поток вычисляет *m*/*p* последовательных элементов результирующего вектора *C*[*m*]
- Если *m* не делится без остатка на *p*, остаток назначается потоку *p* - 1 (равномерная загрузка потоков?)

DGEMV: параллельная версия

```
void run_parallel()
{
    double *a, *b, *c;

    // Allocate memory for 2-d array a[m, n]
    a = xmalloc(sizeof(*a) * m * n);
    b = xmalloc(sizeof(*b) * n);
    c = xmalloc(sizeof(*c) * m);

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++)
            a[i * n + j] = i + j;
    }
    for (int j = 0; j < n; j++)
        b[j] = j;

    double t = wtime();
    matrix_vector_product_omp(a, b, c, m, n);
    t = wtime() - t;

    printf("Elapsed time (parallel): %.6f sec.\n", t);
    free(a);
    free(b);
    free(c);
}
```

DGEMV: параллельная версия

```
int main(int argc, char **argv)
{
    printf("Matrix-vector product (c[m] = a[m, n] * b[n]; m = %d, n = %d)\n", m, n);
    printf("Memory used: %" PRIu64 " MiB\n", ((m * n + m + n) * sizeof(double)) >> 20);

    run_serial();
    run_parallel();

    return 0;
}
```

Анализ эффективности OpenMP-версии

- **Введем обозначения:**
  - $T(n)$ – время выполнения последовательной программы (serial program) при заданном размере *n* входных данных
  - $T_p(n,p)$ – время выполнения параллельной программы (parallel program) с *p* потоками при заданном размере *n* входных данных
- **Коэффициент $S_p(n)$ ускорения** параллельной программ (Speedup):

$$S_p(n) = \frac{T(n)}{T_p(n)}$$

Во сколько раз параллельная программа выполняется на *p* процессорных ядрах быстрее последовательной программы при обработке одних и тех же данных размера *n*

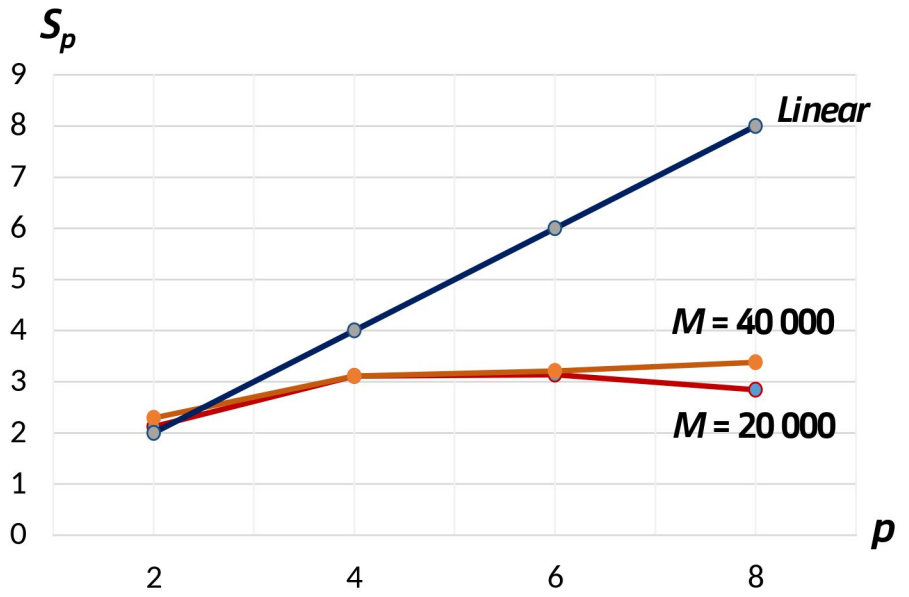
- Как правило

$$S_p(n) \le p$$

- **Цель распараллеливания –**
достичь линейного ускорения на наибольшем числе процессоров: $S_p(n) \ge c \cdot p,$ при $p \to \infty$ и $c > 0$

Анализ эффективности OpenMP-версии

M = N	Количество потоков								
	2			4		6		8	
	T_1	T_2	S_2	T_4	S_4	T_6	S_6	T_8	S_8
20 000 (~ 3 GiB)	0.73	0.34	**2.12**	0.24	**3.11**	0.23	**3.14**	0.25	**2.84**
40 000 (~ 12 GiB)	2.98	1.30	**2.29**	0.95	**3.11**	0.91	**3.21**	0.87	**3.38**
49 000 (~ 18 GiB)								1.23	**3.69**



Вычислительный узел кластера 2 x Intel Quad Xeon E5620:

- **8 ядер (2 x 4 cores)**
- **24 GiB RAM** – 6 x 4GB DDR3 1067 MHz
- CentOS 6.5 x86_64, GCC 4.4.7
- Ключи компиляции: `-std=c99 -Wall -O2 -fopenmp`

Низкая масштабируемость!

Причины ?

DGEMV: конкуренция за доступ к памяти

```
/* matrix_vector_product_omp: Compute matrix-vector product c[m] = a[m][n] * b[n] */
void matrix_vector_product_omp(double *a, double *b, double *c, int m, int n)
{
    #pragma omp parallel
    {
        int nthreads = omp_get_num_threads();
        int threadid = omp_get_thread_num();
        int items_per_thread = m / nthreads;
        int lb = threadid * items_per_thread;
        int ub = (threadid == nthreads - 1) ? (m - 1) : (lb + items_per_thread - 1);

        for (int i = lb; i <= ub; i++) {
            c[i] = 0.0; // Store - запись в память
            for (int j = 0; j < n; j++)
                // 4 обращения к памяти: Load c[i], Load a[i][j], Load b[j], Store c[i]
                // 2 арифметические операции + и *
                c[i] = c[i] + a[i * n + j] * b[j];
        }
    }
}
```

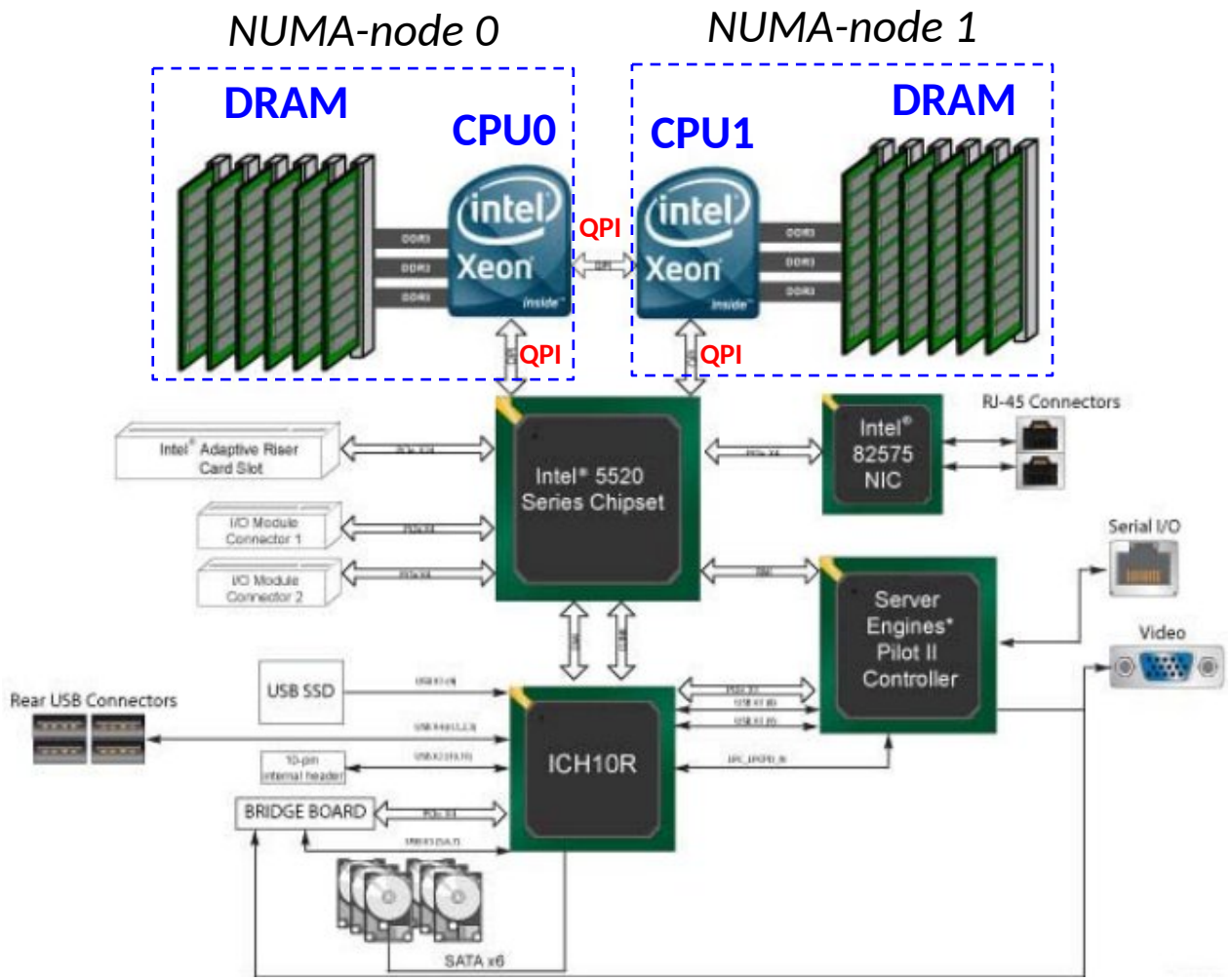
- DGEMV – *data intensive application*
- Конкуренция за доступ к контролеру памяти
- ALU ядер загружены незначительно

Конфигурация узла кластера

Вычислительный узел:

- **System board: Intel 5520UR** (NUMA-система)
- **Процессоры связаны шиной QPI Link: 5.86 GT/s**
- **24 GiB RAM** – 6 x 4GB DDR3 1067 MHz

```
$ numactl --hardware

available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6
node 0 size: 12224 MB
node 0 free: 11443 MB
node 1 cpus: 1 3 5 7
node 1 size: 12288 MB
node 1 free: 11837 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```



http://download.intel.com/support/motherboards/server/s5520ur/sb/e44031012_s5520ur_s5520urt_tps_r1_9.pdf

NUMA (Non-Uniform Memory Access)

- **NUMA-система — многопроцессорная система с общей памятью:**
  - каждый процессор (группа процессоров, ядер) имеют доступ к своей физической памяти через выделенный контроллер
  - доступ к физической памяти других процессоров осуществляется через межпроцессорное соединение

- **NUMA node** — совокупность процессорных ядер и их локальной памяти
  (NUMA-узел может не иметь процессора)

- **Local node** — локальный для ядра NUMA-узел (доступ к памяти через локальный контроллер памяти)

- **Remote node** — удаленный NUMA-узел (доступ к памяти через межпроцессорное соединение + контроллер памяти)

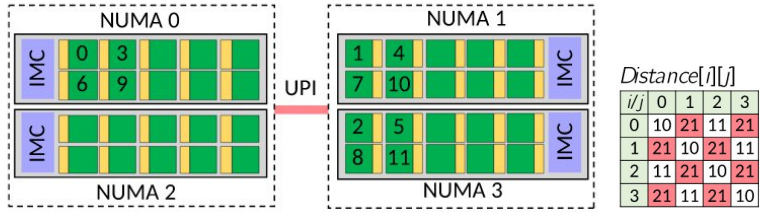
NUMA (Non-Uniform Memory Access)

- **Разбиение на NUMA-узлы программно-настраиваемое (BIOS/UEFI):**
  - один контроллер памяти — один NUMA-узел
  - один контроллер памяти — два NUMA-узла
  - все контроллеры памяти — один NUMA-узел (NUMA-режим отключен, прозрачное чередование доступа к памяти NUMA-узлов)
  - Intel Sub-NUMA Clustering, AMD EPYC Channel-Interleaving, Huawei Channel Interleaving/One Numa Per Socket

- **Межпроцессорное соединение** (Intel UPI, AMD Infinity Fabric, Huawei Hydra) обеспечивает когерентность кеш-памяти процессоров

- Ядро получает информацию о NUMA-топологии из таблицы SRAT (System/Static Resource Affinity Table, ACPI Specification) https://uefi.org/specs/ACPI/6.5/17_NUMA_Architecture_Platforms.html

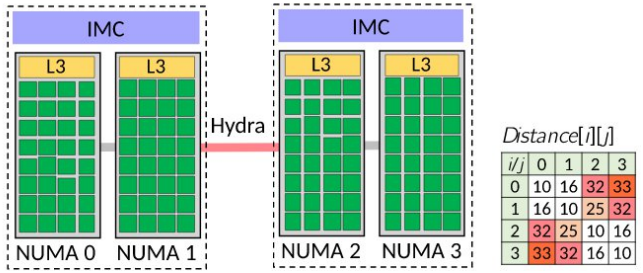
2 x Intel Cascade Lake (20 cores, 2 Sub-NUMA Clusters)



Distance[i][j]

i/j	0	1	2	3
0	10	21	11	21
1	21	10	21	11
2	11	21	10	21
3	21	11	21	10

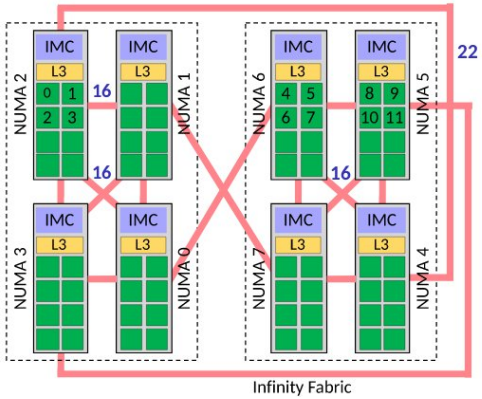
2 x HiSilicon TaiShan v110 (64 cores, 2 NUMA, ARMv8)



Distance[i][j]

i/j	0	1	2	3
0	10	16	32	33
1	16	10	25	32
2	32	25	10	16
3	33	32	16	10

2 x AMD EPYC (32 cores, 4 NUMA)



Distance[i][j]

i/j	0	1	2	3	4	5	6	7
0	10	16	16	16	28	28	22	28
1	16	10	16	16	28	28	28	22
2	16	16	10	16	22	28	28	28
3	16	16	16	10	28	22	28	28
4	28	28	22	28	10	16	16	16
5	28	28	28	22	16	10	16	16
6	22	28	28	28	16	16	10	16
7	28	22	28	28	16	16	16	10

GNU/Linux NUMA

двухпроцессорных сервер, 2 NUMA-узла

```
$ grep NUMA=y /boot/config-`uname -r`
CONFIG_NUMA=y
CONFIG_AMD_NUMA=y
CONFIG_X86_64_ACPI_NUMA=y
CONFIG_ACPI_NUMA=y

$ dmesg | grep -i numa
[    0.000000] NUMA: Initialized distance table, cnt=2
[    0.000000] NUMA: Node 0 [mem 0x00000000-0x7fffffff] + [mem 0x100000000-0x87fffffff] -> [mem 0x00000000-0x87fffffff]
[    0.000000] mempolicy: Enabling automatic NUMA balancing. Configure with numa_balancing= or the kernel.numa_balancing
sysctl
[    0.352790] pci_bus 0000:ff: Unknown NUMA node; performance will be reduced
[    0.361236] pci_bus 0000:7f: Unknown NUMA node; performance will be reduced
[    0.390115] pci_bus 0000:00: on NUMA node 0
[    0.397086] pci_bus 0000:80: on NUMA node 1


$ cat /sys/devices/system/node/possible
0-1

$ cat /sys/devices/system/node/online
0-1

$ lscpu | grep -i numa
  NUMA node(s):          2
  NUMA node0 CPU(s):     0-7,16-23
  NUMA node1 CPU(s):     8-15,24-31
```

GNU/Linux NUMA

Система с одним многоядерным процессором

```
$ grep NUMA=y /boot/config-`uname -r`
CONFIG_NUMA=y
CONFIG_AMD_NUMA=y
CONFIG_X86_64_ACPI_NUMA=y
CONFIG_ACPI_NUMA=y

$ cat /sys/devices/system/node/online
0

$ lscpu | grep -i numa
NUMA node(s):                    1
NUMA node0 CPU(s):               0-7
```

- https://www.kernel.org/doc/html/latest/admin-guide/mm/numa_memory_policy.html
- https://www.kernel.org/doc/Documentation/ABI/stable/sysfs-devices-node
- https://www.kernel.org/doc/html/latest/admin-guide/numastat.html

GNU/Linux NUMA

двухпроцессорных сервер, 2 NUMA-узла

```
$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23
node 0 size: 32072 MB
node 0 free: 31609 MB
node 1 cpus: 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
node 1 size: 32209 MB
node 1 free: 31243 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10

$ numactl --show
policy: default
preferred node: current
physcpubind: 0 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
cpubind: 0 1
nodebind: 0 1
membind: 0 1
```

GNU/Linux NUMA

- Для каждого NUMA-узла ядро поддерживает список физический страниц памяти, зон памяти (memory zone), а также текущую статистику использования ресурсов

```
$ cat /proc/zoneinfo
Node 0, zone      DMA
...
Node 0, zone    DMA32
  pages free     473189
...
Node 0, zone   Normal
  pages free     7615936
...
Node 1, zone      DMA
...
Node 1, zone    DMA32
...
Node 1, zone   Normal
  pages free     7999278
...
```

```
$ cat /sys/devices/system/node/node0/meminfo
Node 0 MemTotal:       32842028 kB
Node 0 MemFree:        32369688 kB
Node 0 MemUsed:          472340 kB
Node 0 SwapCached:            0 kB
...
Node 0 HugePages_Total:     0
Node 0 HugePages_Free:      0
Node 0 HugePages_Surp:      0

$ cat /sys/devices/system/node/node0/vmstat
nr_free_pages 8092422
...
numa_hit 2216296
numa_miss 0
numa_foreign 0
numa_interleave 2021
numa_local 2213164
numa_other 3118
...
```

Политики управления памятью (Linux Memory Policy)

- **Политика памяти (memory policy)** определяет с какого NUMA-узла ядро будет выделять физические страницы память
- **Области действия политики (scope)**
  - System Default Policy — выделение страниц памяти с NUMA-узла локального для ядра, на котором выполняется процесс (local allocation)
  - Task/Process Policy — политика выделения страниц памяти для всего адресного пространства заданного процесса/потока (по умолчанию — System Default Policy)
  - VMA Policy — политика выделения страниц памяти для заданного диапазона адресов адресного пространства процесса
  - Shared Policy — политика выделения страниц памяти для заданной области, которая совместно используется несколькими процессами
- **Режим действия политики (mode)** определяет NUMA-узлы, с которых выделяются страницы памяти
  - Bind — страница памяти выделяется с ближайшего NUMA-узла из заданного множества
  - Preferred — страница памяти выделяется с заданного NUMA-узла, в случае ошибки память выделяется с ближайшего узла с достаточным объемом свободной памяти
  - Interleaved — страницы памяти циклически выделяются с разных NUMA-узлов (чередуются)
  - Preferred Many — страница памяти выделяется NUMA-узла из заданного множества, в случае ошибки память выделяется с ближайшего узла с достаточным объемом свободной памяти

Linux Memory Policy API & CLI

- **Task Memory Policy**
  - `long` **`set_mempolicy`**`(int mode, const unsigned long *nmask, unsigned long maxnode)`
  - `long` **`get_mempolicy`**`(int *mode, const unsigned long *nmask, unsigned long maxnode, void *addr, int flags)`
- **VMA Memory Policy**
  - `long` **`mbind`**`(void *start, unsigned long len, int mode, const unsigned long *nmask, unsigned long maxnode, unsigned flags)`

```
# Выделение страниц памяти с NUMA-узлов 0, 1, привязка процесса к процессорам NUMA-узла 0
$ numactl --membind=0,1 --cpubind=0 ./app

# Чередование выделения страниц памяти с NUMA-узлов 0, 1
$ numactl --interleave=0,1 ./app

# Выделение страниц памяти с локального NUMA-узла
$ numactl --localalloc ./app
```

Linux NUMA-aware Scheduler

- **Планировщик процессов Linux формирует иерархические домены (scheduling domain), соответствующие NUMA-топологии системы**
- Уровни формирования доменов:
  - Уровень аппаратных потоков одного ядра (L1): балансирование загрузки относительно частое, небольшие издержки из-за переноса процесса в пределах группы (имеют общую кеш-память)
  - Уровень ядер одного процессора (L2)**:** балансирование загрузки относительно редко, издержки из-за переноса процесса в пределах домена (на новом ядре нет данных в кеш-памяти)
  - Уровень процессоров NUMA-узла (L3): балансирование загрузки осуществляется редкое, высокие издержки из-за переноса процесса в пределах домена

- Планирование:
  - Процесс запланирован на выполнение: остается на том же процессоре или перемещается на менее загруженный в пределах домена L1
  - Запускается новый процесс (exec()): данных в кеш-памяти нет, планировщик поднимается по иерархии доменов и выбирает наименее загруженный процессор

- Scheduling domains // https://lwn.net/Articles/80911/
- Scheduler domains // https://docs.kernel.org/scheduler/sched-domains.html

Конфигурация узла кластера Oak

Вычислительный узел:

- **System board: Intel 5520UR** (NUMA-система)
- **Процессоры связаны шиной QPI Link: 5.86 GT/s**
- **24 GiB RAM** – 6 x 4GB DDR3 1067 MHz

```
$ numactl --hardware

available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6
node 0 size: 12224 MB
node 0 free: 11443 MB
node 1 cpus: 1 3 5 7
node 1 size: 12288 MB
node 1 free: 11837 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

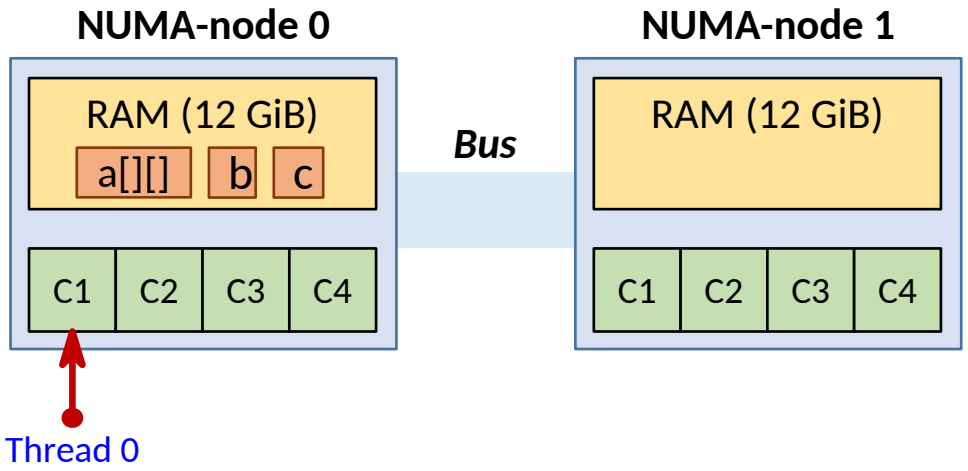
А на каком NUMA-узле (узлах) размещена матрица A и векторы B, C?

http://download.intel.com/support/motherboards/server/s5520ur/sb/e44031012_s5520ur_s5520urt_tps_r1_9.pdf

Выделение памяти потокам в GNU/Linux

- Страница памяти выделяется с NUMA-узла того потока, который первый к ней обратился (*first-touch policy*)
- Данные желательно инициализировать теми потоками, которые будут с ними работать

```
void run_parallel()
{
    double *a, *b, *c;

    // Allocate memory for 2-d array a[m, n]
    a = xmalloc(sizeof(*a) * m * n);
    b = xmalloc(sizeof(*b) * n);
    c = xmalloc(sizeof(*c) * m);

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++)
            a[i * n + j] = i + j;
    }
    for (int j = 0; j < n; j++)
        b[j] = j;
```

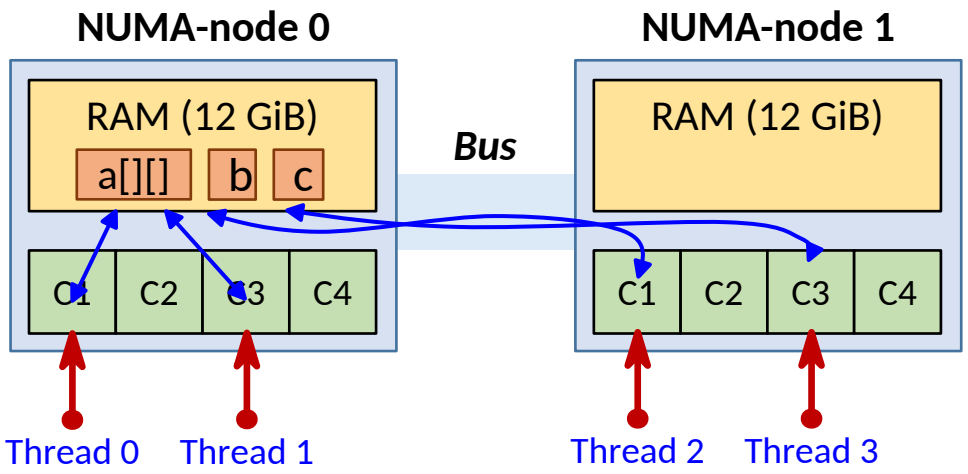


- Поток 0 запрашивает выделение памяти под массивы
- Пока хватает памяти, ядро выделяет страницы с NUMA-узла 0, затем с NUMA-узла 1

Выделение памяти потокам в GNU/Linux

- Страница памяти выделяется с NUMA-узла того потока, который первый к ней обратился (*first-touch policy*)
- Данные желательно инициализировать теми потоками, которые будут с ними работать

```
void run_parallel()
{
    double *a, *b, *c;

    // Allocate memory for 2-d array a[m, n]
    a = xmalloc(sizeof(*a) * m * n);
    b = xmalloc(sizeof(*b) * n);
    c = xmalloc(sizeof(*c) * m);

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++)
            a[i * n + j] = i + j;
    }
    for (int j = 0; j < n; j++)
        b[j] = j;
```



- Обращение к массивам из потоков NUMA-узла 1 будет идти через межпроцессорную шину в память узла 0

Параллельная инициализация массивов

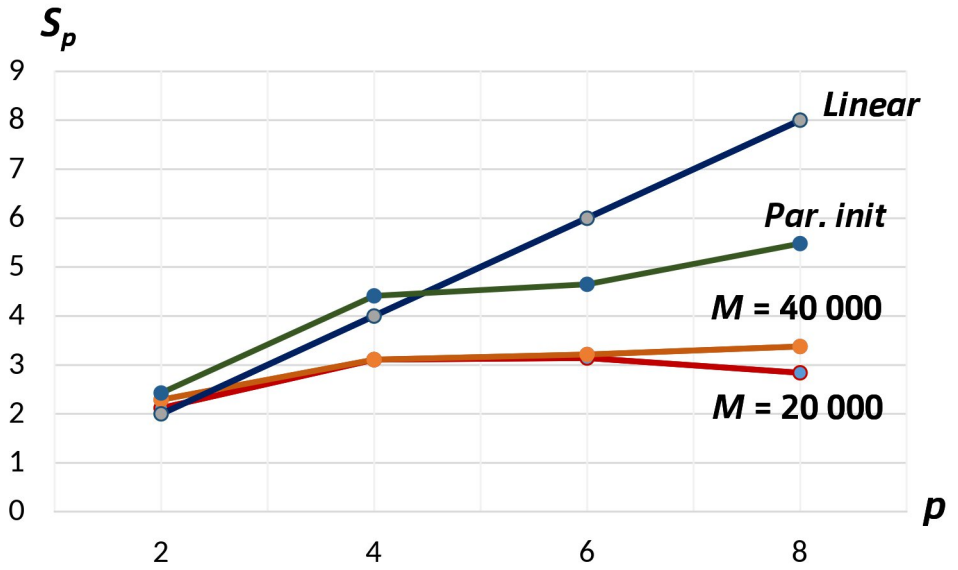
```
void run_parallel()
{
    // Allocate memory
    double *a = xmalloc(sizeof(*a) * m * n);
    double *b = xmalloc(sizeof(*b) * n);
    double *c = xmalloc(sizeof(*c) * m);

    #pragma omp parallel
    {
        int nthreads = omp_get_num_threads();
        int threadid = omp_get_thread_num();
        int items_per_thread = m / nthreads;
        int lb = threadid * items_per_thread;
        int ub = (threadid == nthreads - 1) ? (m - 1) : (lb + items_per_thread - 1);

        for (int i = lb; i <= ub; i++) {
            for (int j = 0; j < n; j++)
                a[i * n + j] = i + j;     // Страницы памяти выделяются с NUMA-узла потока, который
            c[i] = 0.0;                   // инициализировал элемент массива
        }
    }
    for (int j = 0; j < n; j++)
        b[j] = j;

    /* ... */
}
```

Анализ эффективности OpenMP-версии (2)

M = N	Количество потоков								
	2			4		6		8	
	T_1	T_2	S_2	T_4	S_4	T_6	S_6	T_8	S_8
20 000 (~ 3 GiB)	0.73	0.34	**2.12**	0.24	**3.11**	0.23	**3.14**	0.25	**2.84**
40 000 (~ 12 GiB)	2.98	1.30	**2.29**	0.95	**3.11**	0.91	**3.21**	0.87	**3.38**
49 000 (~ 18 GiB)								1.23	3.69

Parallel initialization									
40 000 (~ 12 GiB)	2.98	1.22	**2.43**	0.67	**4.41**	0.65	**4.65**	0.54	**5.48**
49 000 (~ 18 GiB)								0.83	**5.41**

Суперлинейное ускорение (super-linear speedup): $S_p(n) > p$



Улучшили масштабируемость

Дальнейшие оптимизации:

- Эффективный доступ к кеш-памяти
- Векторизация кода (SSE/AVX)
- …

Automatic NUMA Balancing

- Ядро Linux отслеживает и периодически перемещает страницы виртуальной памяти на NUMA-узел, с которого к ним часто обращаются процессы

```
$ cat /proc/sys/kernel/numa_balancing
1

# Disable Automatic NUMA Balancing (current session)
echo 0 > /proc/sys/kernel/numa_balancing

# Disable Automatic NUMA Balancing
$ sysctl -w kernel.numa_balancing=0
```

Automatic NUMA Balancing

- Ядро периодически сканирует адресное пространство процесса и помечает часть страниц памяти для срабатывания обработчика при следующем обращении к ним (NUMA hinting fault — NHF)
- При срабатывании обработчика NHF ядру становится известен источник запроса — процесс и его NUMA-узел
- Страница переносится на другой NUMA-узел если к ней обращаются дважды с одного и того же NUMA-узла (shared fault) или дважды обращается один и тот же процесс (private fault)

- Многопоточные приложения:
  - Псевдослучайное многократное обращение к произвольным страницам из всех потоков: страницы мигрируют между NUMA-узлами
  - Каждый поток обращается к своей области памяти (группе страниц): минимум переносов страниц

```
$ sysctl -a | grep numa_balancing
kernel.numa_balancing_scan_delay_ms = 1000        # Задержка перед первым сканирование памяти процесса
kernel.numa_balancing_scan_period_max_ms = 60000  # Макс. время сканирования памяти процесса
kernel.numa_balancing_scan_period_min_ms = 1000   # Мин. время сканирования памяти процесса
kernel.numa_balancing_scan_size_mb = 256          # Размер окна сканирования для одного прохода
```

- Rik van Riel. **Automatic NUMA Balancing** // https://events.static.linuxfound.org/sites/events/files/slides/summit2014_riel_chegu_w_0340_automatic_numa_balancing_0.pdf

Статистика выделения страниц памяти с NUMA-узла

```
$ cat /sys/devices/system/node/node0/numastat
numa_hit 3425512
numa_miss 0
numa_foreign 0
interleave_hit 2021
local_node 3421949
other_node 3505
```

```
$ cat /sys/devices/system/node/node1/numastat
numa_hit 6822476
numa_miss 0
numa_foreign 0
interleave_hit 1000
local_node 6814006
other_node 8434
```

numa_hit	Процесс запрашивает выделение страницы памяти с этого узла, запрос успешно удовлетворяется
numa_miss	Процесс запрашивает выделение памяти с другого узла, но память выделятся с текущего узла
numa_foreign	Процесс запрашивает выделение памяти с этого узла, но память выделяется с другого узла
local_node	Процесс запущен процессоре этого узла и память выделена с этого узла
other_node	Процесс запущен процессоре другого узла, но память выделена с этого узла
interleave_hit	При чередовании запрашивается выделение памяти с текущего узла, запрос успешно удовлетворяется

- Если запрос процесса на выделение памяти с предпочитаемого узла удовлетворен, то numa_hit увеличивается на предпочитаемом узле
- В противном случае, на предпочитаемом узле увеличивается numa_foreign, а на узле с которого выделили память увеличивается numa_miss

https://docs.kernel.org/admin-guide/numastat.html

Сводная статистика выделения страниц памяти с NUMA-узлов

```
$ grep -i numa /proc/vmstat
numa_hit 10206402
numa_miss 0
numa_foreign 0
numa_interleave 3021
numa_local 10194405
numa_other 11903

numa_pte_updates 350494             # Количество страниц отмеченных для проверки доступа (NHF)
numa_huge_pte_updates 0             # Количество больших страниц отмеченных для проверки доступа (NHF)
numa_hint_faults 314809             # Количество NHF
numa_hint_faults_local 151721       # Количество NHF с локальных узлов
numa_pages_migrated 62982           # Количество перемещенных страниц (все узлы, все приложения)
```

Автоматический перенос страниц памяти (Automatic NUMA Balancing)

`kernel.numa_balancing=1`

```
$ ./dgemv_omp
# NREPS = 1
/proc/vmstat: numa_pages_migrated 924689
DGEMV C=A*B (c[m] = a[m, n] * b[n]; m = 10000, n = 10000)
Memory used: 763 MiB
Master thread NUMA node: 1 (cpu 24)
Elapsed time (sec): 0.064324;  3109.27 GFLOPS

 Performance counter stats for './dgemv_omp':
                 0      cpu-migrations:u

/proc/vmstat: numa_pages_migrated 924689
```

0
4K pages migrated
(16 threads, 2 NUMA nodes)

```
$ ./dgemv_omp
# NREPS = 100
/proc/vmstat: numa_pages_migrated 924689
DGEMV C=A*B (c[m] = a[m, n] * b[n]; m = 10000, n = 10000)
Memory used: 763 MiB
Master thread NUMA node: 1 (cpu 24)
Elapsed time (sec): 0.053285;  3753.42 GFLOPS

 Performance counter stats for './dgemv_omp'
                 0      cpu-migrations:u

/proc/vmstat: numa_pages_migrated 1022354
```

97 665
4K pages migrated
(16 threads, 2 NUMA nodes)

```
void run_parallel()
{
    double *a, *b, *c;
    a = malloc(sizeof(*a) * M * N);
    b = malloc(sizeof(*b) * N);
    c = malloc(sizeof(*c) * M);
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++)
            a[i * N + j] = i + j;
    }
    for (int j = 0; j < N; j++)
        b[j] = j;

    int NREPS = 100;  // Многократный проход по массиву
    double t = wtime();
    for (int i = 0; i < NREPS; i++)
      dgemv_omp(a, b, c, M, N);
    t = wtime() - t;
    t /= NREPS;

    double gflops = M * N * 2 * 1E-6 / t;
    printf(«Elapsed time (sec): %.6f;  %.2f GFLOPS\n»,
           t, gflops);

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

- За несколько проходов по массивам a, b, c включается балансировщик и страницы переносятся на другой NUMA-узел

Автоматический перенос страниц памяти (Automatic NUMA Balancing)

`kernel.numa_balancing=1`

```
OMP_NUM_THREADS=16 ./dgemv_omp    # NREPS=1000
```

```
$ sudo numatop
*** Monitoring 394 processes and 480 threads (interval: 5.0s)
   PID            PROC     RMA(K)     LMA(K)    RMA/LMA        CPI     *CPU%%
 75634       dgemv_omp     5509.5     3474.8        1.6       0.58        6.9
 75621         numatop       10.7       74.0        0.1       1.07        0.0
  1341        telegraf      107.0       52.7        2.0       1.59        0.0
  1110     systemd-oom       13.0       11.6        1.1       3.78        0.0
...

*** Monitoring 390 processes and 476 threads (interval: 5.0s)
   PID            PROC     RMA(K)     LMA(K)    RMA/LMA        CPI     *CPU%%
 75634       dgemv_omp    16415.3    43396.5        0.4       0.49       41.6
 79577         numatop       10.7       60.8        0.2       1.12        0.1
  1341        telegraf      141.4      121.3        1.2       1.66        0.1
 79462     kworker/0:2       27.8       26.7        1.0       2.90        0.0
  1110     systemd-oom       17.4       12.5        1.4       3.66        0.0
```

- Часть потоков обращаются к памяти удаленного NUMA-узла
- Обращений к памяти удаленного узла больше на 60% (RMA/LMA=1.6)

- Страницы памяти перенесены
- Доминируют обращения к памяти локальных узлов

- RMA — Remote Memory Access
- LMA — Local Memory Access

`https://github.com/intel/numatop`
`http://01.org/numatop`

- Автоматическая миграция страниц положительно влияет на производительность приложений, многократно обращающихся к одним и тем же страницам памяти
- Если dgemv_omp() вызывается один раз, балансировщик не включается
- Если dgemv_omp() вызывается многократно, то страницы с данными массива a[][] перемещаются на другие NUMA-узлы, повышается коэффициент локальных обращений

Автоматический перенос страниц памяти (Automatic NUMA Balancing)

`kernel.numa_balancing=1`

```
OMP_NUM_THREADS=16 ./dgemv_omp_init
```

```
$ sudo numatop
*** Monitoring 393 processes and 479 threads (interval: 5.0s)
   PID            PROC     RMA(K)     LMA(K)    RMA/LMA         CPI     *CPU%%
 79877    dgemv_omp_i      627.7     5576.8        0.1        0.68        8.7
  1341       telegraf       70.0      117.8        0.6        2.77        0.1
 79864        numatop       10.3       18.7        0.6        1.42        0.0
  1110    systemd-oom        7.1        5.6        1.3        4.16        0.0
...

*** Monitoring 393 processes and 479 threads (interval: 5.0s)
   PID            PROC     RMA(K)     LMA(K)    RMA/LMA         CPI     *CPU%%
 79877    dgemv_omp_i     2420.6    19563.9        0.1        0.66       32.2
 79864        numatop       15.8       70.4        0.2        1.06        0.1
 79858    kworker/0:0       56.4       57.6        1.0        3.57        0.0
 79666    kworker/u65       47.5       49.0        1.0        3.09        0.0
  1110    systemd-oom       18.7       15.6        1.2        3.14        0.0
```

- Инициализация потоками своих областей матрицы a[][] обеспечивает выделение страниц памяти с локальных NUMA-узлов
- В версии dgemv_omp_init доминируют обращения к памяти локальных NUMA-узлов (RMA/LMA=0.1)

Отключение автоматического переноса страниц памяти

`kernel.numa_balancing=0`

`OMP_NUM_THREADS=16 ./dgemv_omp`

```
$ sudo numatop
*** Monitoring 387 processes and 473 threads (interval: 5.0s)
   PID            PROC     RMA(K)     LMA(K)    RMA/LMA         CPI     *CPU%%
 76078       dgemv_omp    11784.2     5720.1        2.1        0.55        9.9
  1341        telegraf       95.0       51.4        1.9        1.47        0.0
 76065         numatop        8.3       22.8        0.4        1.18        0.0
 76077            perf        2.0       62.6        0.0        0.80        0.0
...

*** Monitoring 387 processes and 473 threads (interval: 5.0s)
   PID            PROC     RMA(K)     LMA(K)    RMA/LMA         CPI     *CPU%%
 76078       dgemv_omp    31918.4    34492.6        0.9        0.53       42.1
 76065         numatop       13.7      115.8        0.1        1.16        0.2
 75519     kworker/u65       43.9       46.4        0.9        3.29        0.0
 76018     kworker/0:1       23.0       20.1        1.1        2.93        0.0
  1110     systemd-oom       17.5       13.3        1.3        4.54        0.0
```

Отключение автоматического переноса страниц памяти

`kernel.numa_balancing=0`

`OMP_NUM_THREADS=16 ./dgemv_omp_init`

```
$ sudo numatop
*** Monitoring 389 processes and 475 threads (interval: 5.0s)
   PID            PROC      RMA(K)      LMA(K)    RMA/LMA         CPI      *CPU%%
 76208     dgemv_omp_i       264.3     35127.9         0.0        0.49        25.7
 76195         numatop         8.5        33.2         0.3        1.18         0.0
  1183       sssd_sudo        69.7        68.8         1.0        2.68         0.0
  1110     systemd-oom        11.1         9.2         1.2        4.02         0.0
    16     ksoftirqd/0        19.6        20.0         1.0        2.85         0.0
 76156     kworker/0:0        16.7        16.8         1.0        2.77         0.0
 75981     kworker/u65         4.4         8.2         0.5        3.79         0.0
```

M=N=20000, 16 threads, 2 x Intel Xeon E5-2620 v4 (8 cores), linux 5.18.11

NUMA Balancing	dgemv_omp		dgemv_omp_init	
	Time (sec)	GFLOPS	Time (sec)	GFLOPS
NUMA Balancing off	0.248783	3215.65	0.183197	4366.88
NUMA Balancing on	0.248147	3223.89	0.183158	**4367.82**