

Семинар 1

Стандарт OpenMP (часть 1)

Михаил Курносов

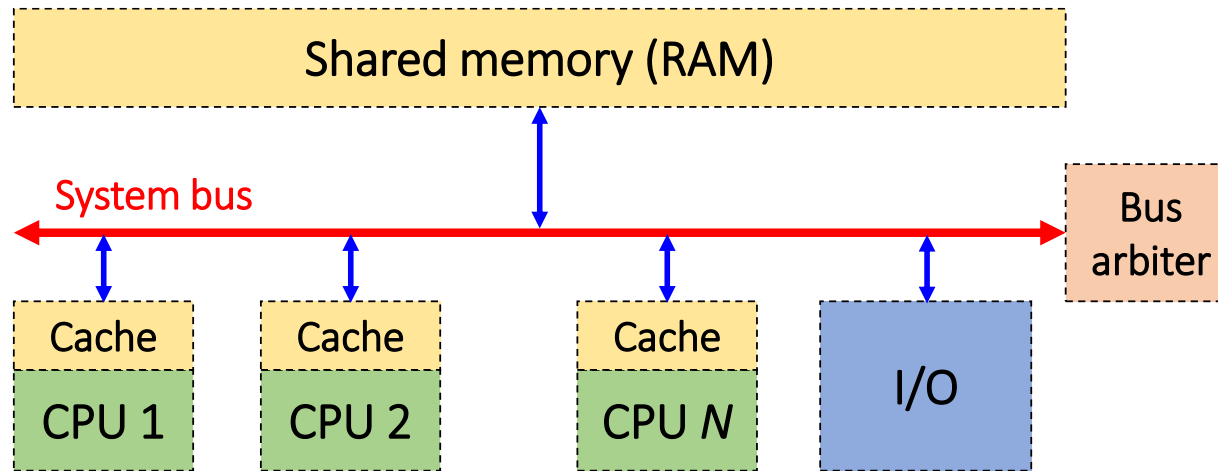
E-mail: mkurnosov@gmail.com

WWW: www.mkurnosov.net

Цикл семинаров «Основы параллельного программирования»
Институт физики полупроводников им. А. В. Ржанова СО РАН
Новосибирск, 2015

Многопроцессорные системы с общей памятью

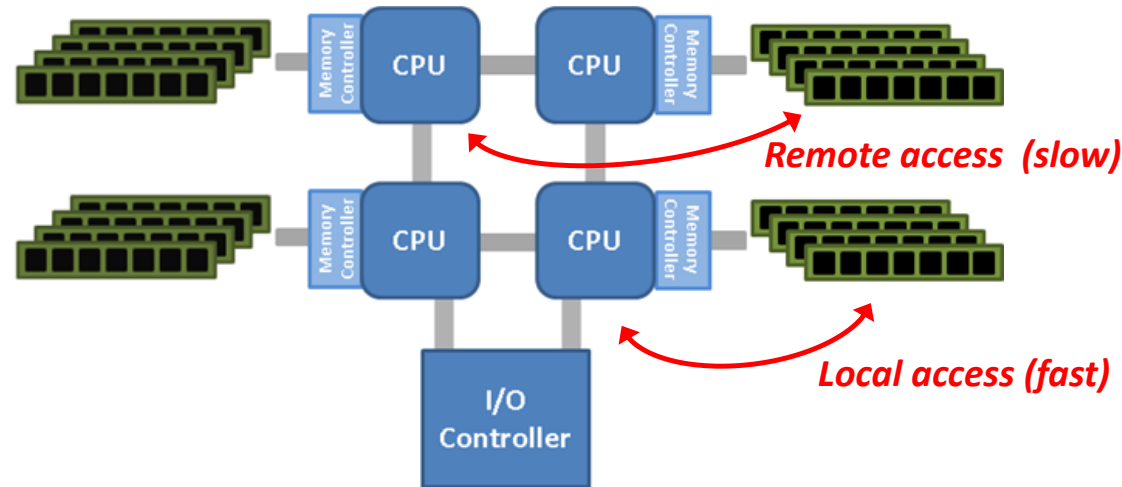
SMP-системы



- **Процессоры SMP-системы** имеют одинаковое время доступа к разделяемой памяти: симметричный, однородный доступ к памяти – Uniform Memory Access (UMA)
- **Системная шина** (system bus, crossbar) – узкое место, ограничивающее производительность вычислительного узла (процессоры конкурируют за доступ к ней)
- **Проблемы:** низкая масштабируемость, обеспечение когерентности (согласованности) кеш-памяти процессоров и разделяемой памяти

Многопроцессорные системы с общей памятью

NUMA-системы



- **NUMA** (Non-Uniform Memory Architecture) – многопроцессорная вычислительная система с неоднородным доступом к разделяемой памяти
- Процессоры сгруппированы в **NUMA-узлы** со своей локальной памятью (NUMA nodes)
- **Разное время доступа** к локальной памяти NUMA-узла (NUMA-node) и памяти другого узла
- Большая масштабируемость по сравнению с SMP-системами
- **Проблемы:** обеспечение когерентности кеш-памяти процессоров

Процессы и потоки



```
// Uninitialized data (BSS)
int sum[100]; // BSS

// Initialized data (Data)
float grid[100][100] = {1.0};

int main()
{
    // Local variable (stack)
    double s = 0.0;

    // Allocate from the heap
    float *x = malloc(1000);
    // ...
    free(x);
}
```

Процессы и потоки



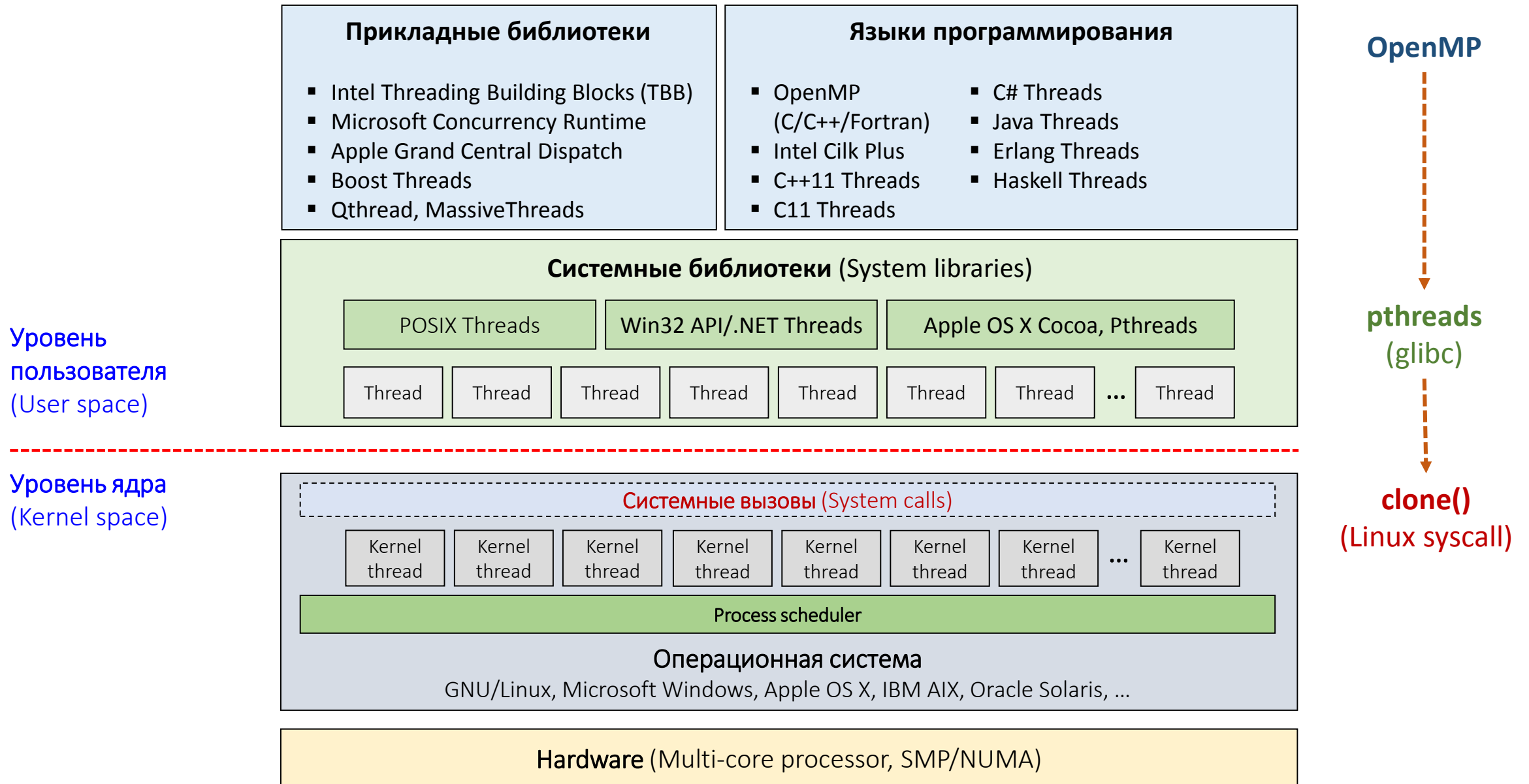
```
// Uninitialized data (BSS)
int sum[100]; // BSS

// Initialized data (Data)
float grid[100][100] = {1.0};

int main()
{
    // Local variable (stack)
    double s = 0.0;

    // Allocate from the heap
    float *x = malloc(1000);
    // ...
    free(x);
}
```

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 3.1 (2011), **OpenMP 4.0 (2013)**
 - ❑ <http://www.openmp.org>
 - ❑ <http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf>
- Требуется поддержка со стороны компилятора



Поддержка компиляторами

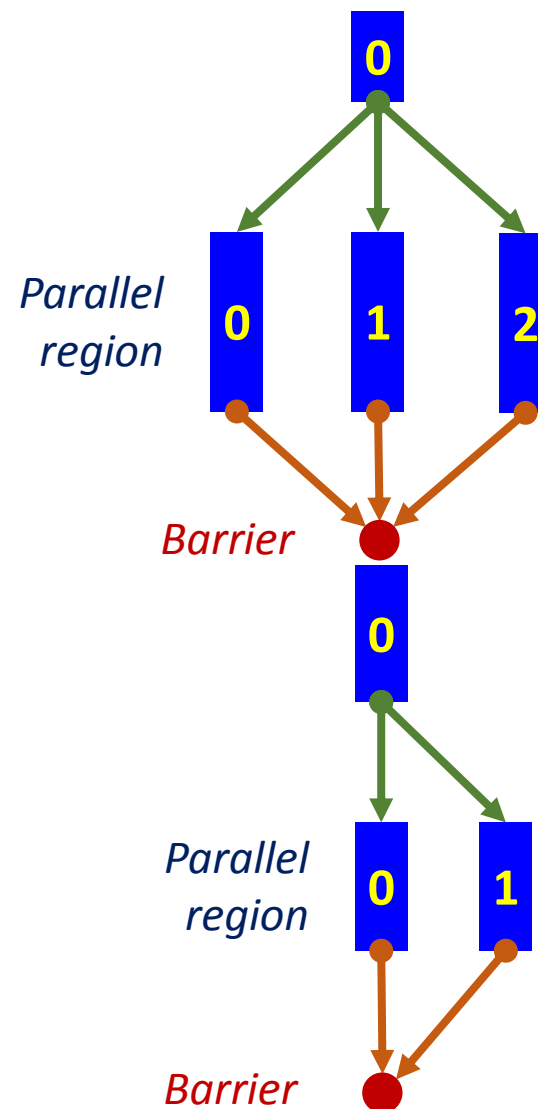
Compiler	Information
GNU GCC	Option: <code>-fopenmp</code> gcc 4.2 – OpenMP 2.5, gcc 4.4 – OpenMP 3.0, gcc 4.7 – OpenMP 3.1 gcc 4.9 – OpenMP 4.0
Clang (LLVM)	OpenMP 3.1 clang + Intel OpenMP RTL http://clang-omp.github.io/
Intel C/C++, Fortran	OpenMP 4.0 Option: <code>-Qopenmp</code> , <code>-openmp</code>
Oracle Solaris Studio C/C++/Fortran	OpenMP 4.0 Option: <code>-xopenmp</code>
Microsoft Visual Studio C++	Option: <code>/openmp</code> OpenMP 2.0 only
Other compilers: IBM XL, PathScale, PGI, Absoft Pro, ...	

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



Hello, World

```
#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

```
#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

Умножение матрицы на вектор (DGEMV)

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

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

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

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

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];  
    }  
}
```

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

DGEMV: последовательная версия

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

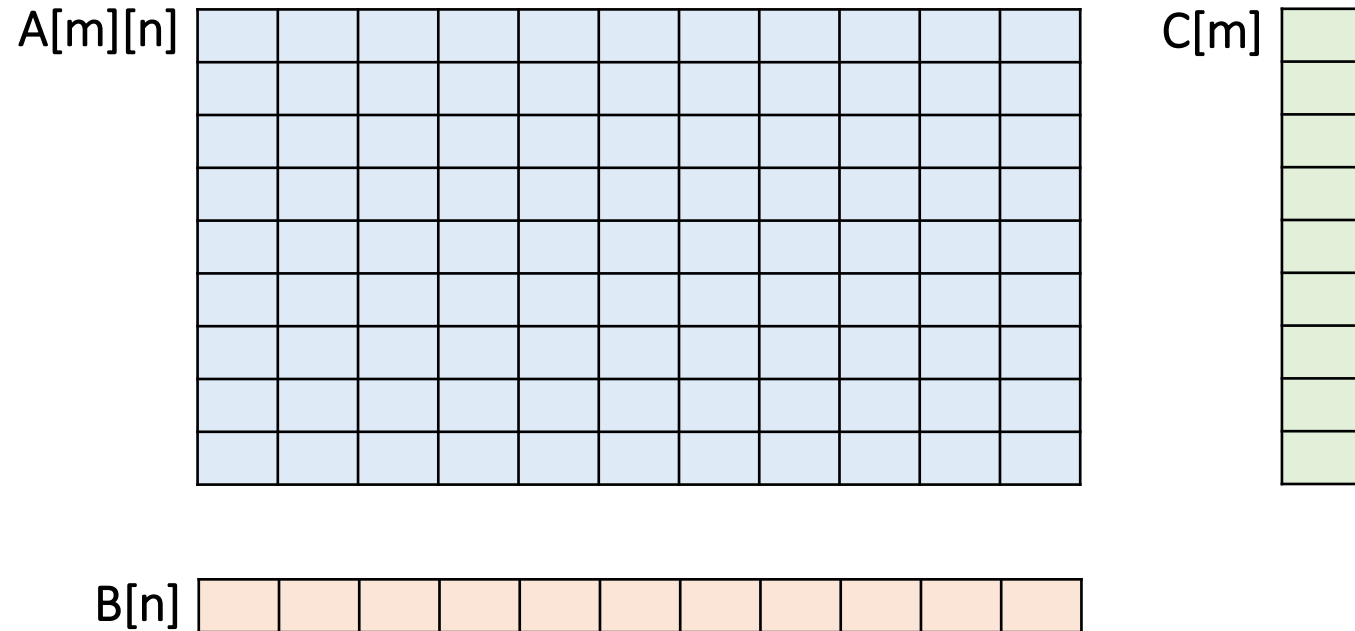
    a = xmalloc(sizeof(*a) * m * n);
    b = xmalloc(sizeof(*b) * n);
    c = xmalloc(sizeof(*c) * m);

    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

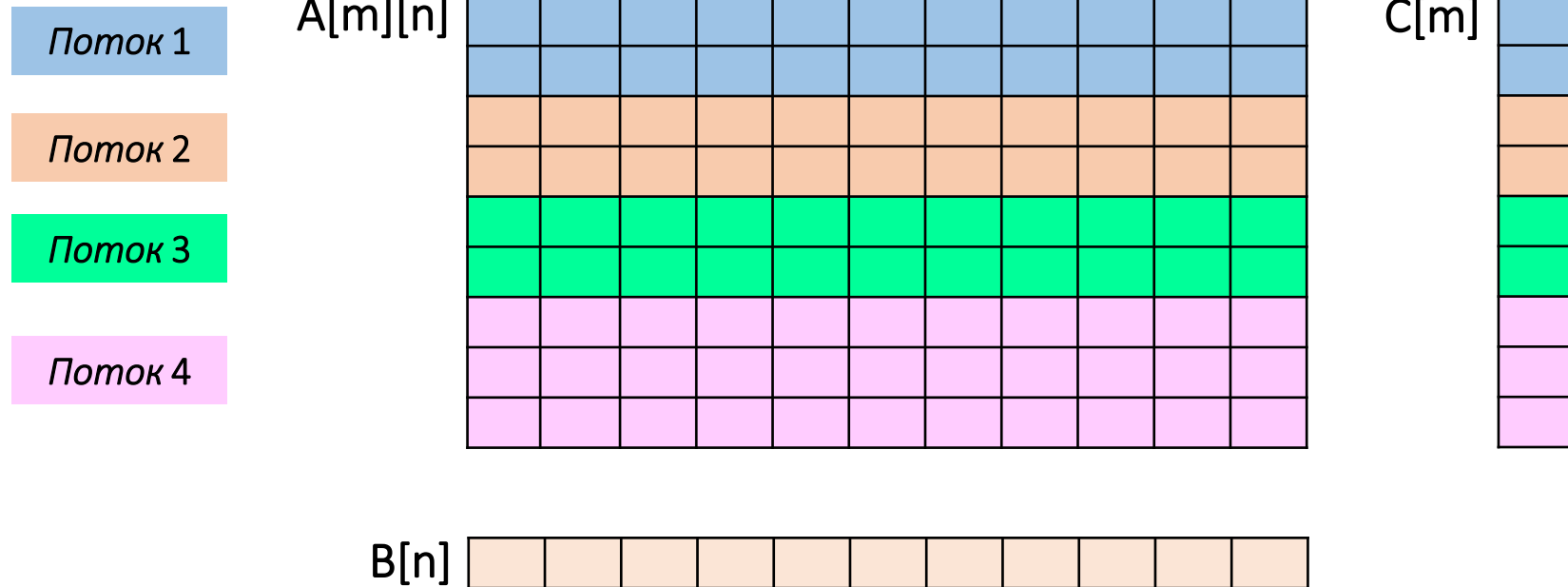


```
for (int i = 0; i < m; i++) {  
    c[i] = 0.0;  
    for (int j = 0; j < n; j++)  
        c[i] += a[i * n + j] * b[j];  
}
```

Требования к параллельному алгоритму

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

DGEMV: параллельная версия



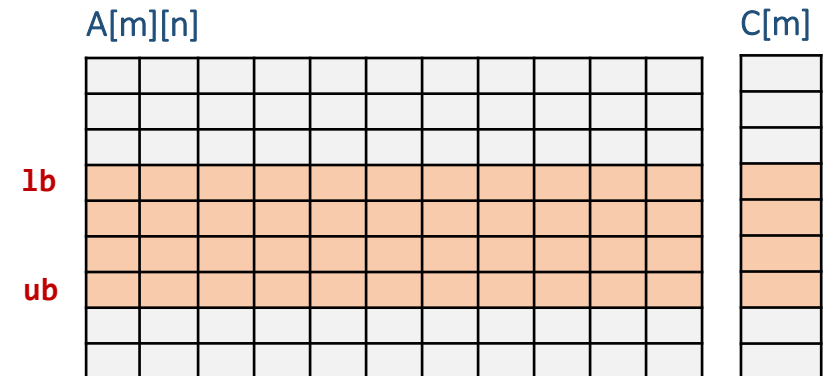
```
for (int i = 0; i < m; i++) {  
    c[i] = 0.0;  
    for (int j = 0; j < n; j++)  
        c[i] += a[i * n + j] * b[j];  
}
```

Распараллеливание внешнего цикла

- Каждому потоку выделяется часть строк матрицы A

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];  
        }  
    }  
}
```



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) на p процессорах 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 процессорах быстрее последовательной программы при обработке одних и тех же данных размера n

- Как правило

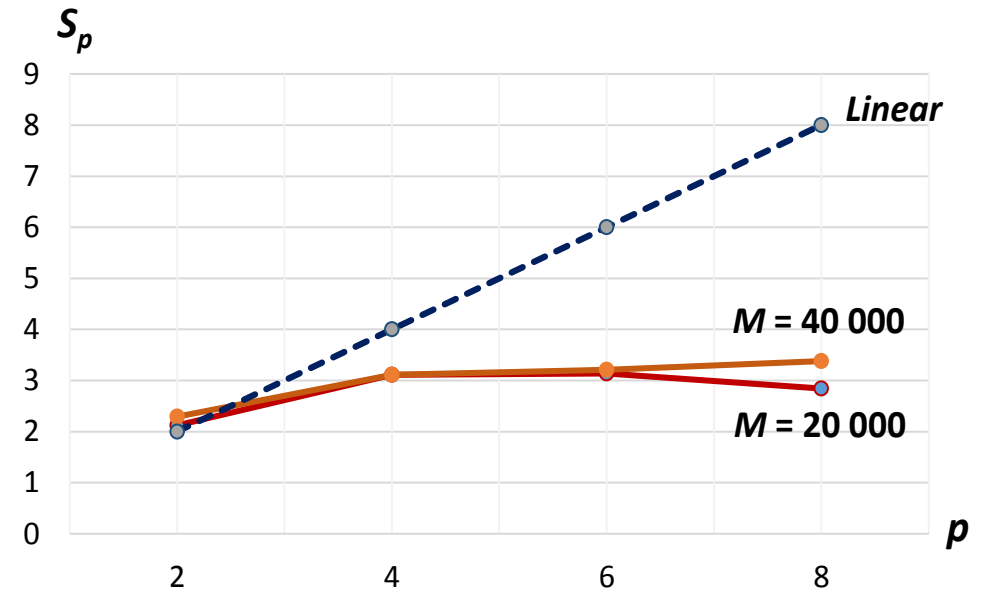
$$S_p(n) \leq p$$

- Цель распараллеливания –

достичь линейного ускорения на наибольшем числе процессоров: $S_p(n) \geq c \cdot p$, при $p \rightarrow \infty$ 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 Oak (oak.crpt.sibsutis.ru):

- System board: Intel 5520UR
- 8 ядер – два Intel Quad Xeon E5620 (2.4 GHz)
- 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++)  
                // Memory ops: Load c[i], Load a[i][j], Load b[j], Store c[i]  
                c[i] += a[i * n + j] * b[j];  
        }  
    }  
}
```

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

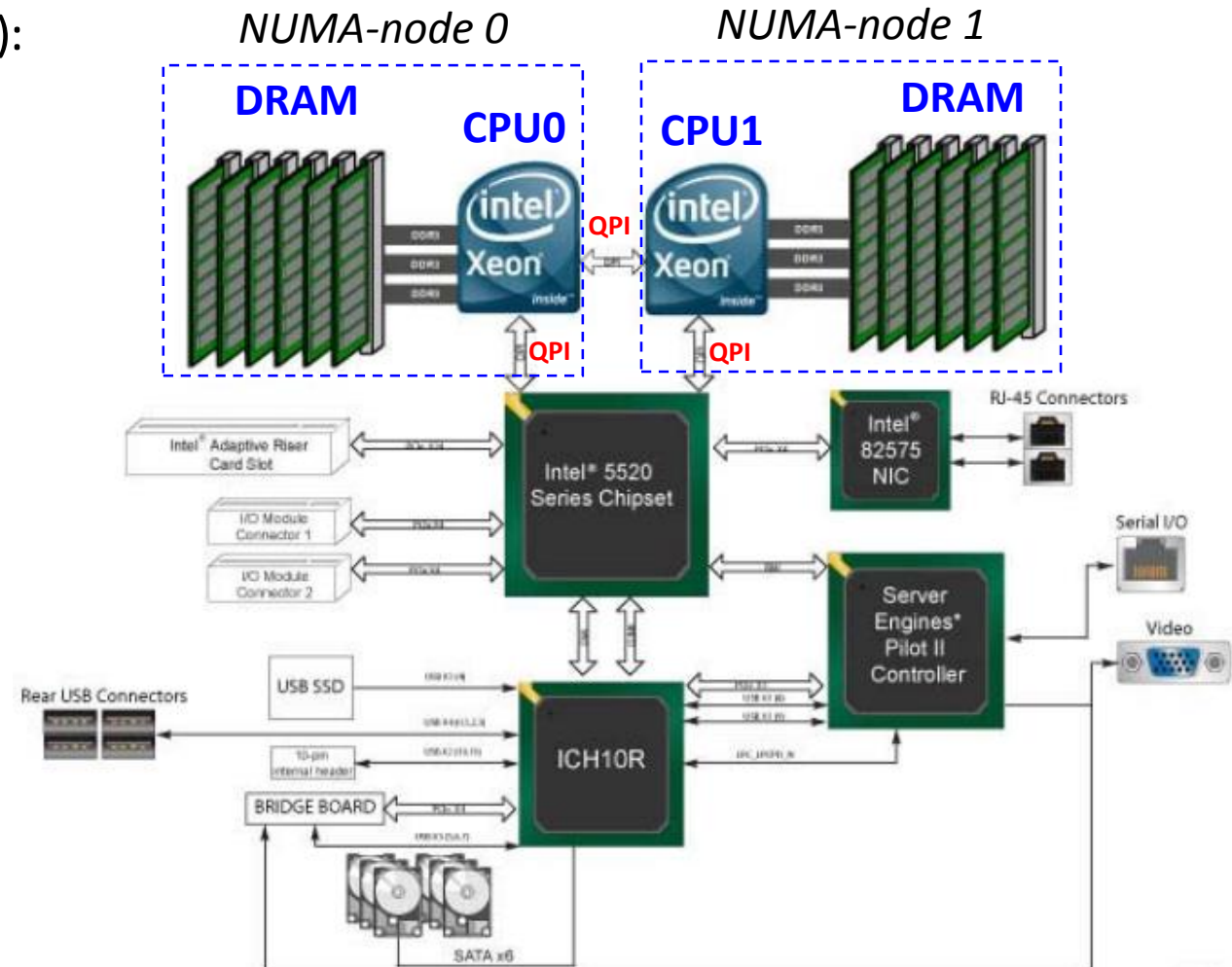
Конфигурация узла кластера Oak

Вычислительный узел кластера Oak (oak.crpt.sibsutis.ru):

- 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
```



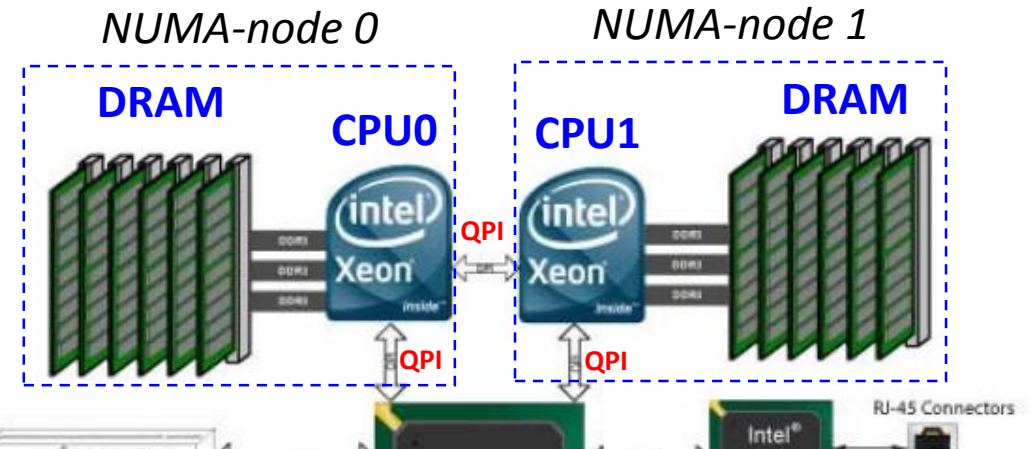
Конфигурация узла кластера Oak

Вычислительный узел кластера Oak (oak.crpt.sibsutis.ru):

- 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
```

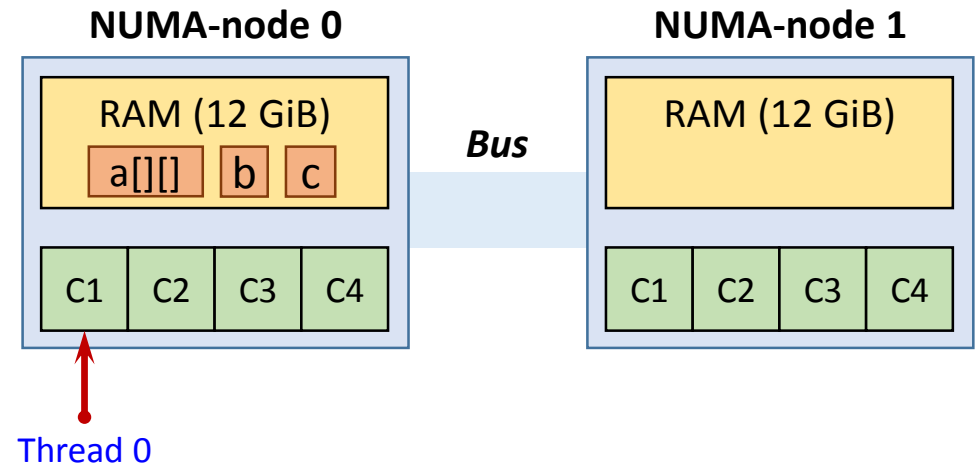


А на каком NUMA-узле (узлах)
размещена матрица A и векторы B, C?

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;  
}
```

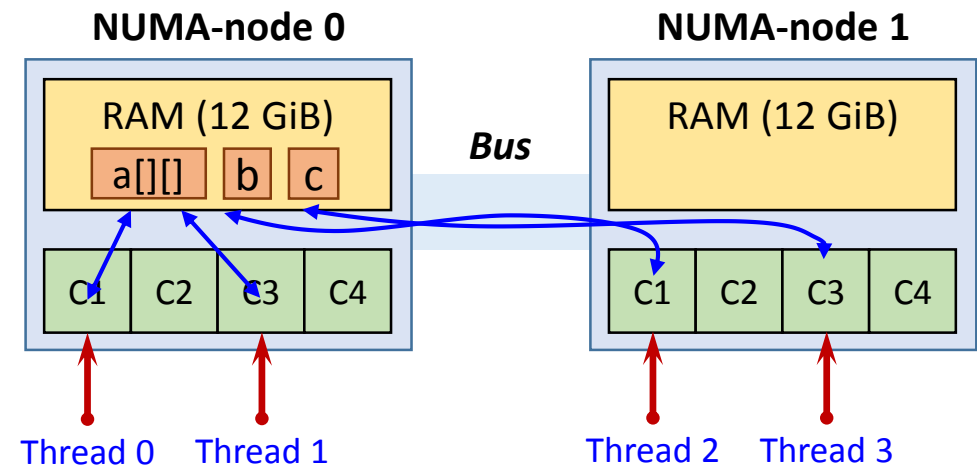


- Поток 0 запрашивает выделение памяти под массивы
- Пока хватает памяти, ядро выделяет страницы с NUMA-узла 0, затем с NUMA-узла 1

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()
{
    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);

    #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;
            c[i] = 0.0;
        }
    }
    for (int j = 0; j < n; j++)
        b[j] = j;

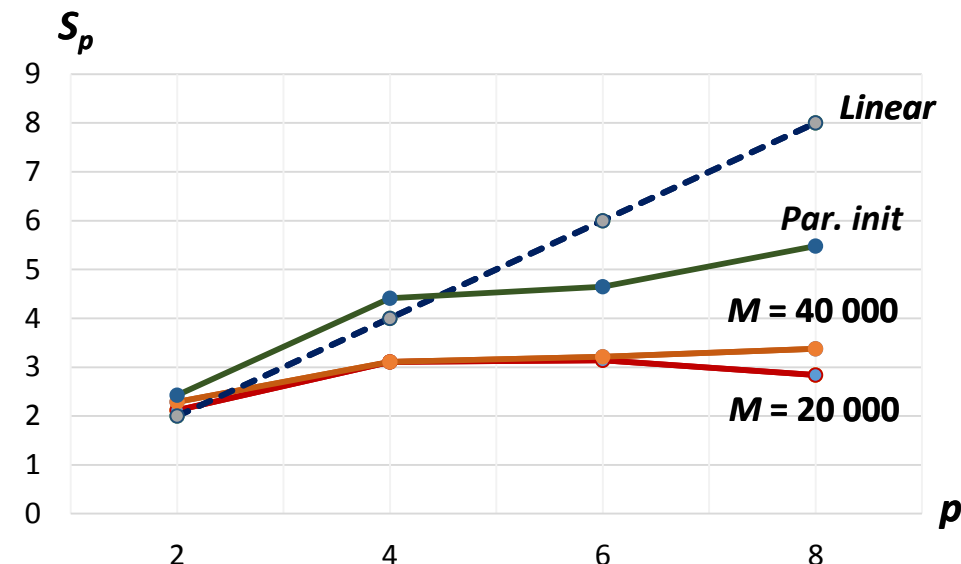
    /* ... */
}
```


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)
- ...

Подключение к кластеру oak.crct.sibsutis.ru

```
$ ssh mkurnosov@oak.cpct.sibsutis.ru
```

Password:

```
[mkurnosov@oak ~]$ mc
```

```
Left      File      Command  Options  Right
<- ~ .[~]> <- /home/pub .[~]>
'n      Name      Size      Permission  'n      Name      Size      Modify time
/..      UP--DIR  drwxr-xr-x
/.cache  4096     drwx-----
/.dbus   4096     drwx-----
/.gconf  4096     drwx-----
/.gconfd 4096     drwx-----
/.gnome2 4096     drwxr-xr-x
/.gnome2_private 4096     drwx-----
/.mc     4096     drwx-----
/.mozilla 4096     drwxr-xr-x
/.ssh    4096     drwx-----
/temp    4096     drwxr-xr-x
.Xauthority 115     -rw-----
.bash_history 9059    -rw-----
.bash_logout 18      -rw-r--r--
.bash_profile 176     -rw-r--r--
.bashrc   189     -rw-r--r--
.emacs    500     -rw-r--r--

UP--DIR 413G/458G (90%)
UP--DIR 413G/458G (90%)

Hint: Leap to frequently used directories in a single bound with C-\.
```

Запуск OpenMP-программ на oak.cpct.sibsutis.ru

Компилируем программу

\$ make

gcc -std=c99 -g -Wall -O2 -fopenmp -c matvec.c -o matvec.o

gcc -o matvec matvec.o -fopenmp

Ставим задание в очередь системы SLURM

\$ sbatch ./task-slurm.job

Submitted batch job 3609

Проверяем состояние очереди задач

\$ squeue

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
3609	debug	task-slu	ivanov	R	0:01	1	cn2

Открываем файл с результатом

\$ cat ./slurm-3609.out

#!/bin/bash

#SBATCH --nodes=1 --ntasks-per-node=8

export OMP_NUM_THREADS=2

./matvec

Пользователь

Состояние

Узел кластера

Спасибо за внимание!

Условная компиляция

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

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

#ifdef _OPENMP
    #pragma omp parallel num_threads(6)
    {
        printf("Hello, multithreaded world: thread %d of %d\n",
               omp_get_thread_num(), omp_get_num_threads());
    }
    printf("OpenMP version %d\n", _OPENMP);

    if (_OPENMP >= 201107)
        printf("  OpenMP 3.1 is supported\n");
#endif

    return 0;
}
```

Условная компиляция

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

```
$ ./hello
```

```
Hello, multithreaded world: thread 2 of 4
```

```
Hello, multithreaded world: thread 1 of 4
```

```
Hello, multithreaded world: thread 0 of 4
```

```
Hello, multithreaded world: thread 3 of 4
```

```
OpenMP version 201107
```

```
OpenMP 3.1 is supported
```

Условная компиляция

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

```
$ ./hello
```


Время выполнения отдельных потоков

```
void matrix_vector_product_omp(double *a, double *b, double *c, int m, int n)
{
    #pragma omp parallel
    {
        double t = omp_get_wtime();
        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);

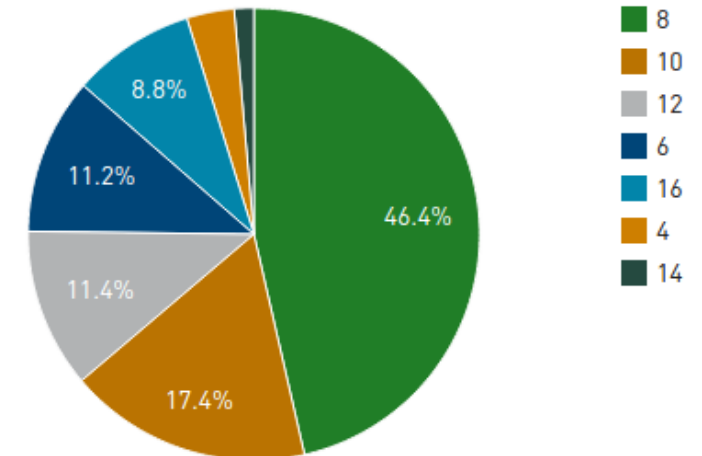
        for (int i = lb; i <= ub; i++) {
            c[i] = 0.0;
            for (int j = 0; j < n; j++)
                c[i] += a[i * n + j] * b[j];
        }
        t = omp_get_wtime() - t;

        printf("Thread %d items %d [%d - %d], time: %.6f\n", threadid, ub - lb + 1, lb, ub, t);
    }
}
```

Многопроцессорные узлы Top500 (2014)

- Среднее количество ядер на сокет (процессор):
9.2 (4, 6, 8, 10, 12, 14, 16)
- Процессоры:
Intel (> 80%), IBM Power, AMD Opteron, SPARC64, ShenWei, NEC
- Ускорители (12% систем):
Intel Xeon Phi, NVIDIA GPU, ATI/AMD GPU
- Операционная система:
GNU/Linux (96%), IBM AIX (11 шт.), Microsoft (2 шт.)

Cores per Socket System Share



www.top500.org

Мультиархитектура современных ВС

- Уровень одного узла (общая память)
 - ✓ Многопоточное программирование (intra-node): [OpenMP](#), Intel TBB/Cilk Plus, C11/C++11 Threads
 - ✓ Программирование ускорителей (NVIDIA/AMD GPU, Intel Xeon Phi): [NVIDIA CUDA](#), OpenCL, OpenACC, OpenMP 4.0
- Множество вычислительных узлов (распределенная память):
 - ✓ [MPI](#), Shmem, PGAS (Cray Chapel, IBM X10), Coarray Fortran, Global Arrays, ...
- Уровень ядра процессора
 - ✓ Vectorization (SIMD: SSE/AVX, AltiVec), cache optimization, superscalar optimizations

