Семинар 2. Введение в многопоточное программирование

Параллелизм в рамках узла. Многопоточность.

Прикладные библиотеки

- Intel Threading Building Blocks (TBB)
- Microsoft Concurrency Runtime
- Apple Grand Central Dispatch
- Boost Threads
- Qthread, MassiveThreads

Языки программирования

- OpenMP (C/C++/Fortran)
- Intel Cilk Plus
- C++11 Threads
- C11 Threads
- C# Threads
- Java Threads
- Erlang Threads
- Haskell Threads

Системные библиотеки (System libraries)

POSIX Threads | Win32 API/.NET Threads | Apple OS X Cocoa, Pthreads

Thread | Thread | Thread | Thread | Thread | Thread | Thread | ... | Thread

Уровень пользователя (User space)

Уровень ядра (Kernel space)

Системные вызовы (System calls)

Kernel thread | Kernel thread | Kernel thread | Kernel thread | Kernel thread | Kernel thread | Kernel thread | ... | Kernel thread

Process scheduler

Операционная система

GNU/Linux, Microsoft Windows, Apple OS X, IBM AIX, Oracle Solaris, ...

Hardware (Multi-core processor, SMP/NUMA)

OpenMP

pthreads (glibc)

clone() (Linux syscall)

https://www.mkurnosov.net/teaching/docs/pct-spring-lec5.pdf

Системные вызовы. Разделение ресурсов.

Создание процессах-потомка посредством копирования с разделением ресурсов:

```
#include <unistd.h>
pid_t fork(void);
```

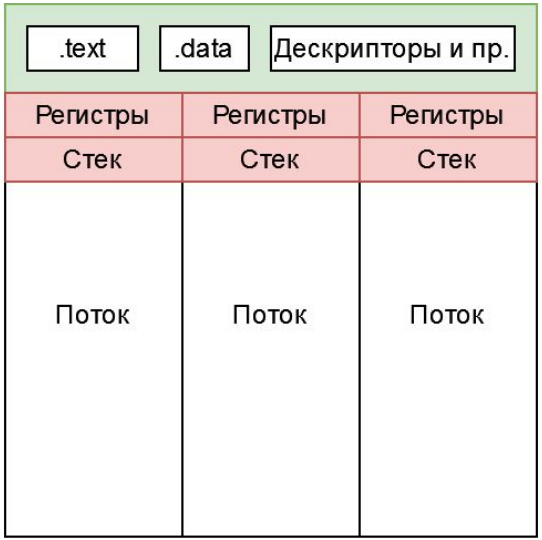
Создание потомка с совместным использованием указанных ресурсов:

```
        /* Prototype for the glibc wrapper function */
        #define _GNU_SOURCE
        #include <sched.h>
        int clone(int (*fn)(void *), void *stack, int flags, void
*arg, ...
                  /* pid_t *parent_tid, void *tls, pid_t
*child_tid */ );

        /* Prototype of the raw clone() system call, see NOTES */
        #include <linux/sched.h>    /* Definition of struct
clone_args */
        #include <sched.h>          /* Definition of CLONE_*
constants */
```

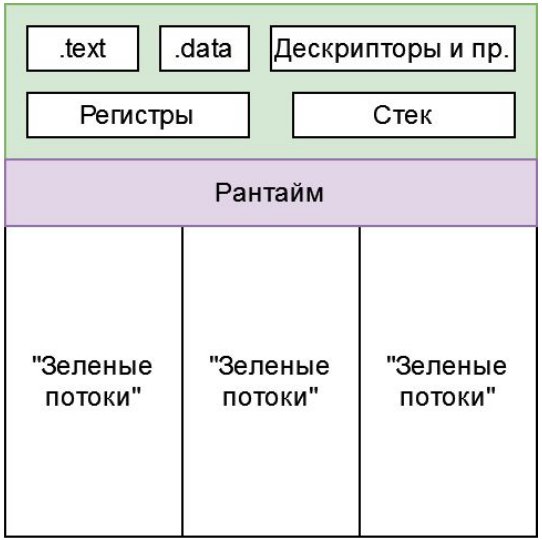
Разделение ресурсов.

Процесс

Нативные потоки



Легкие потоки, сопрограммы и пр.



Потоки POSIX.

pthread_t – идентификатор потока;
pthread_attr_t – перечень атрибутов потока

```
#include <pthread.h>
int pthread_create(pthread_t *thread, const
pthread_attr_t *attr,
    void *(*start)(void *), void *arg);

int pthread_join (pthread_t thread, void ** data);

int pthread_detach(pthread_t thread);

int pthread_cancel(pthread_t thread);
```

Поддержка многопоточности в современном C++.

cppreference.com

Create account

Page Discussion

C++ Concurrency support library

Concurrency support library (since C++11)

C++ includes built-in support for threads, atomic operations, mutual exclusion, condition variabl

Threads

Threads enable programs to execute across several processor cores.

Defined in header <thread>

thread (C++11)	manages a separate thread (class)
jthread (C++20)	std::thread with support for auto-joining and cancellation (class)

Functions managing the current thread

Defined in namespace this_thread

yield (C++11)	suggests that the implementation reschedule execution of threads (function)
get_id (C++11)	returns the thread id of the current thread (function)
sleep_for (C++11)	stops the execution of the current thread for a specified time duration (function)
sleep_until (C++11)	stops the execution of the current thread until a specified time point (function)

```cpp
#include <iostream>
#include <thread>

// N N+1 N+2 ... M

void thread_function(int N, int M, int64_t *sum)
{
    std::cout << "Thread num " << std::this_thread::get_id() << " sum
before is " << *sum << " N =" << N <<"; M = " << M << std::endl;
    for (auto i = N; i < M; ++i)
    {
        *sum += i;
    }
    std::cout << "Thread num " << std::this_thread::get_id() << " sum
after is " << *sum << std::endl;
}
```

```cpp
//#define multiThread
int main()
{
    const int maxVal = 300000000;
#ifdef multiThread
    const int range1 = maxVal / 3;
    const int range2 = 2 * range1;
    int64_t sum1 = 0, sum2 = 0, sum3 = 0;

    std::thread tmpThread1(thread_function, 0, range1, &sum1);
    std::thread tmpThread2(thread_function, range1, range2,
&sum2);
    std::thread tmpThread3(thread_function, range2, maxVal,
&sum3);

    tmpThread1.join();
    tmpThread2.join();
    tmpThread3.join();
    std::cout << "sum is " << sum1 + sum2 + sum3 << std::endl;

#else
    int64_t sum = 0;
    thread_function(0, maxVal, &sum);
    std::cout << "sum is " << sum << std::endl;
#endif
    return 0;
}
```

POSIX mutex

```
pthread_mutex_t

int pthread_mutex_init(pthread_mutex_t *mutex,
const pthread_mutexattr_t *attr);

int pthread_mutex_destroy(pthread_mutex_t *mutex);


int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

OpenMP - высокоуровневый API для многопоточного программирования на C, C++ и Fortran

- OpenMP 1.0 - октябрь 1997
- OpenMP 5.1 - ноябрь 2020
- интерфейс для распараллеливания на CPU/GPU
- поддерживается компиляторами GCC, LLVM, Intel, Nvidia (PGI), IBM

https://www.openmp.org/

1 Overview of the OpenMP API

The collection of compiler directives, library routines, and environment variables that this document describes collectively define the specification of the OpenMP Application Program Interface (OpenMP API) for parallelism in C, C++ and Fortran programs.

This specification provides a model for parallel programming that is portable across architectures from different vendors. Compilers from numerous vendors support the OpenMP API. More information about the OpenMP API can be found at the following web site

https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-1.pdf

Как это выглядит?

- распараллеливание осуществляется с помощью директив компилятора:

```
#pragma omp <OpenMP directive>
```

- область действия директивы - блок кода { … }

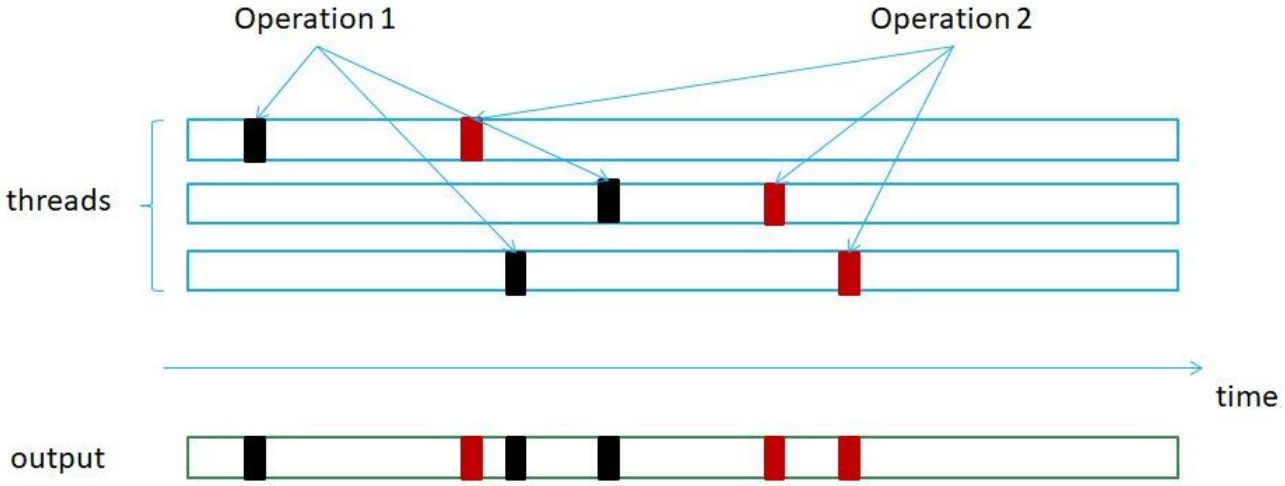
```c
#include <omp.h>
#include <time.h>
#include <stdio.h>

int main()
{
    int i;

    #pragma omp parallel
    {
        printf("Hello world\n");
        printf("OMP thread id: %d\n", omp_get_thread_num());
    }

    return 0;
}
```

```
mkhalilov@MacBook-Air-Arina omp % gcc-11 -fopenmp -O3 test1.c
gcc-11 -fopenmp -O3 test1.c
mkhalilov@MacBook-Air-Arina omp % OMP_NUM_THREADS=3 ./a.out
OMP_NUM_THREADS=3 ./a.out
Hello world
Hello world
OMP thread id: 0
Hello world
OMP thread id: 2
OMP thread id: 1
```

Пример. OpenMP Hello World.

(не)последовательность исполнения в OpenMP

```
mkhalilov@MacBook-Air-Arina omp % gcc-11 -fopenmp -O3 test1.c
gcc-11 -fopenmp -O3 test1.c
mkhalilov@MacBook-Air-Arina omp % OMP_NUM_THREADS=3 ./a.out
OMP_NUM_THREADS=3 ./a.out
Hello world
Hello world
OMP thread id: 0
Hello world
OMP thread id: 2
OMP thread id: 1
```



Наиболее используемый функционал OpenMP

OpenMP pragma, function, or clause	Concepts
#pragma omp parallel	Parallel region, teams of threads, structured block, interleaved execution across threads.
void omp_set_thread_num() int omp_get_thread_num() int omp_get_num_threads()	Default number of threads and internal control variables. SPMD pattern: Create threads with a parallel region and split up the work using the number of threads and the thread ID.
double omp_get_wtime()	Speedup and Amdahl's law. False sharing and other performance issues.
setenv OMP_NUM_THREADS N	Setting the internal control variable for the default number of threads with an environment variable
#pragma omp barrier #pragma omp critical	Synchronization and race conditions. Revisit interleaved execution.
#pragma omp for #pragma omp parallel for	Worksharing, parallel loops, loop carried dependencies.
reduction(op:list)	Reductions of values across a team of threads.
schedule (static [,chunk]) schedule(dynamic [,chunk])	Loop schedules, loop overheads, and load balance.
shared(list), private(list), firstprivate(list)	Data environment.
nowait	Disabling implied barriers on workshare constructs, the high cost of barriers, and the flush concept (but not the flush directive).
#pragma omp single	Workshare with a single thread.
#pragma omp task #pragma omp taskwait	Tasks including the data environment for tasks.

Tim Mattson, The OpenMP Common Core: A hands on exploration, Intel Corp.

Пример 1. Последовательная сумма массива (редукция).

```
17 int main()
18 {
19     double array[ELEMS];
20     double start, end;
21     double sum = 0.0;
22     size_t i, nthreads;
23
24     init_array(array, ELEMS);
25
26     start = omp_get_wtime();
27     for (i = 0; i < ELEMS; i++) {
28         sum += array[i];
29     }
30     end = omp_get_wtime();
31
32     printf("Array sum: %f\n", sum);
33     printf("Elapsed time: %lfs\n", end - start);
34
35     return 0;
36 }
```

Пример 2. Редукция с parallel for.

```c
int main()
{
    double array[ELEMS];
    double start, end;
    double *sum = NULL;
    size_t i, nthreads;

    sum = malloc(sizeof(double) * omp_get_max_threads());
    if (sum == NULL) {
        fprintf(stderr, "Malloc failed. Abort!");
        exit(EXIT_FAILURE);
    }

    init_array(array, ELEMS);

    start = omp_get_wtime();
    #pragma omp parallel
    {
        int nthrds = omp_get_num_threads();
        int tid = omp_get_thread_num();
        int j;

        if (tid == 0)
            nthreads = nthrds;

        for (j = tid, sum[tid] = 0.0;
             j < ELEMS;
             j = j + nthrds) {
            sum[tid] += array[j];
        }
    }

    for (i = 1; i < nthreads; i++)
        sum[0] += sum[i];

    end = omp_get_wtime();

    printf("Array sum: %f\n", sum[0]);
    printf("Elapsed time: %lfs\n", end - start);

    free(sum);
    return 0;
}
```

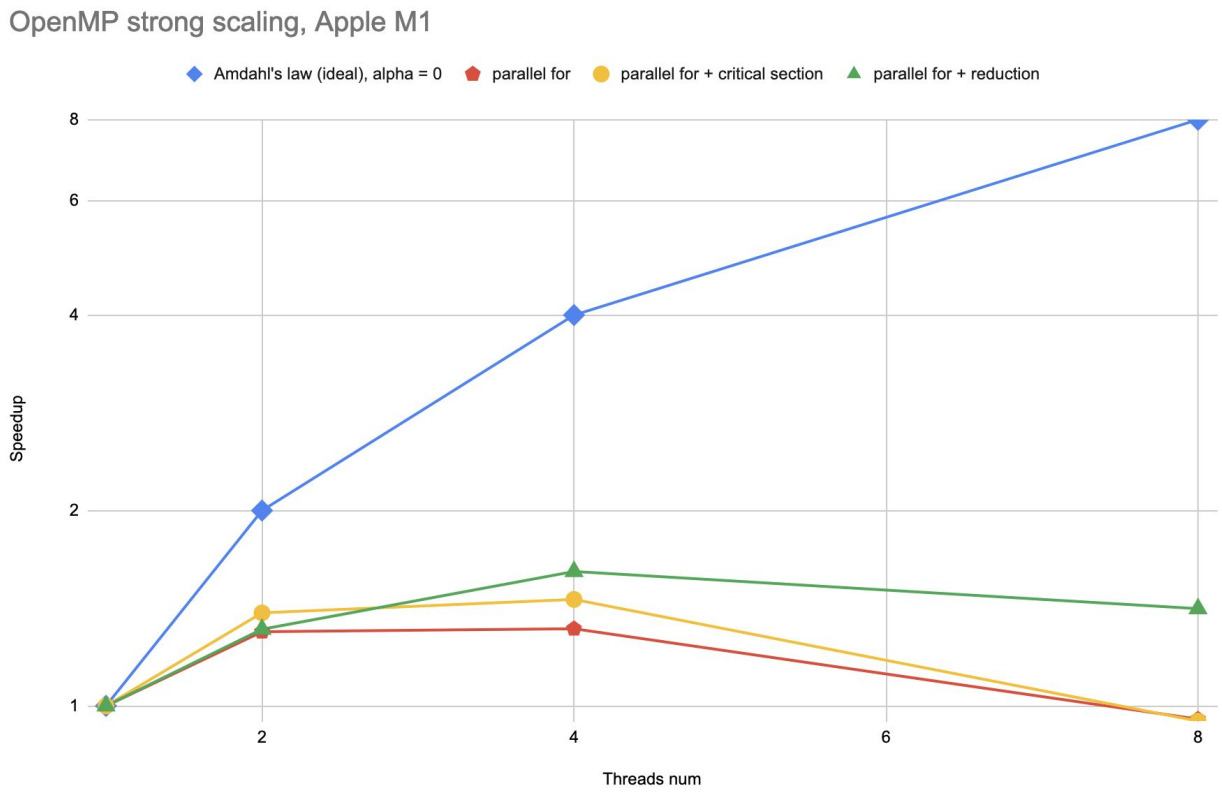
Пример 3. Редукция с parallel for и критической секцией.

```
17 int main()
18 {
19     double start, end;
20     double array[ELEMS];
21     double sum = 0.0;
22     size_t i;
23
24     init_array(array, ELEMS);
25
26     start = omp_get_wtime();
27     #pragma omp parallel
28     {
29         double local_sum = 0.0;
30         int nthrds = omp_get_num_threads();
31         int idx = omp_get_thread_num();
32         int j;
33
34         for (j = idx; j < ELEMS; j = j + nthrds)
35             local_sum += array[j];
36
37         #pragma omp critical
38         sum += local_sum;
39     }
40     end = omp_get_wtime();
41
42     printf("Array sum: %f\n", sum);
43     printf("Elapsed time: %lfs\n", end - start);
44
45     return 0;
46 }
```

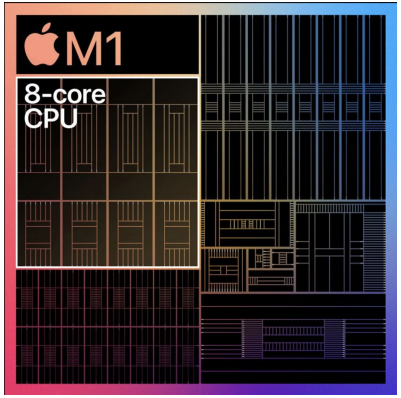
Пример 4. Редукция с omp parallel for и reduction.

```
17 int main()
18 {
19     double array[ELEMS];
20     double start, end;
21     double sum = 0.0;
22     size_t i, nthreads;
23
24     init_array(array, ELEMS);
25
26     start = omp_get_wtime();
27     #pragma omp parallel
28     {
29         #pragma omp for reduction(+:sum)
30         for (i = 0; i < ELEMS; i++) {
31             sum += array[i];
32         }
33     }
34     end = omp_get_wtime();
35
36     printf("Array sum: %f\n", sum);
37     printf("Elapsed time: %lfs\n", end - start);
38
39     return 0;
40 }
```

Сравнение производительности. Закон Амдала. Масштабируемость.



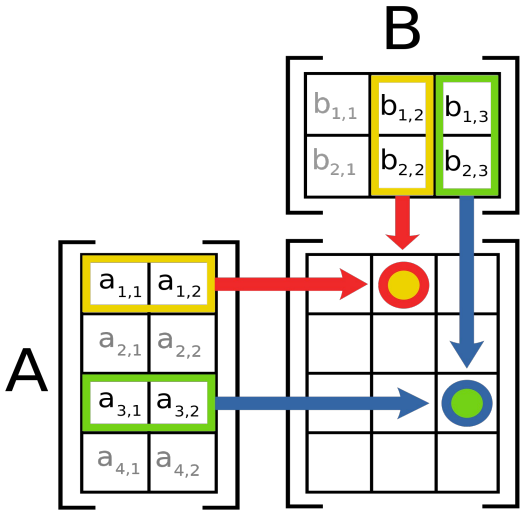
$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$



Домашнее Задание 1. Параллельное умножение матриц (DGEMM)

1. Реализовать параллельную реализацию умножения матриц DGEMM (**D**ouble precision **Ge**neral **M**atrix **M**ultiplication) из пакета BLAS с использованием OpenMP. (4 балла)
2. Привести анализ сильной/слабой масштабируемости параллельной реализации на суперкомпьютере Харизма. (2 балла)
3. Реализовать оптимизированную под узлы суперкомпьютера Харизма параллельную реализацию DGEMM, проанализировать характеристики ее сильной/слабой масштабируемости (2 балла).
4. Реализовать вычисление суммы ряда с использованием pthreads и поддержкой произвольного количества потоков.

$$C = \alpha * A * B + \beta * C$$



https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

Домашнее Задание 1. Замечания.

$$C = \alpha * A * B + \beta * C$$

- alpha = 1 и beta = 0;
- элементы A, B и C имеют тип `double`;
- A, B и C хранятся в одномерных массивах в column-major порядке (https://en.wikipedia.org/wiki/Row-_and_column-major_order);
- функция умножения матриц должна реализовывать следующий интерфейс:

```
void blas_dgemm(int M, int N, int K, double *A, double *B, double *C)
{
    /* My DGEMM implementation goes here */
}
```

Домашнее Задание 1. Замечания.

- ДЗ1 должно быть реализовано на языке C;
- код должен быть оформлен в соответствии с Linux kernel conding style https://www.kernel.org/doc/html/v4.10/process/coding-style.html (**за невыполнение этого критерия оценка за ДЗ1 будет снижена на 3 балла!**);
- комментарии должны **объяснять** неочевидные моменты (если таковые имеются), **а не дублировать написанный код**;
- комментарии должны приводиться **на английском языке;**
- на сдачу ДЗ1 дается вторая попытка при неудовлетворительной первой попытке;
- дедлайн по ДЗ1 будет объявлен на Семинаре 3.

Литература

1) OpenMP API Specification: Version 5.1 November 2020, https://www.openmp.org/spec-html/5.1/openmp.html
2) OpenMP Application Programming Interface Examples, https://www.openmp.org/wp-content/uploads/openmp-examples-5.1.pdf
3) Tim Mattson, The OpenMP Common Core: A hands on exploration, https://extremecomputingtraining.anl.gov/files/2019/07/ATPESC_2019_Track-2_2_7-31_830am_Mattson-The-OpenMP_Common_Core.pdf
4) https://www.mkurnosov.net/teaching/docs/pct-spring-lec5.pdf
5) M. Herlihy, N. Shavit, The Art of Multiprocessor Programming, Revised pring