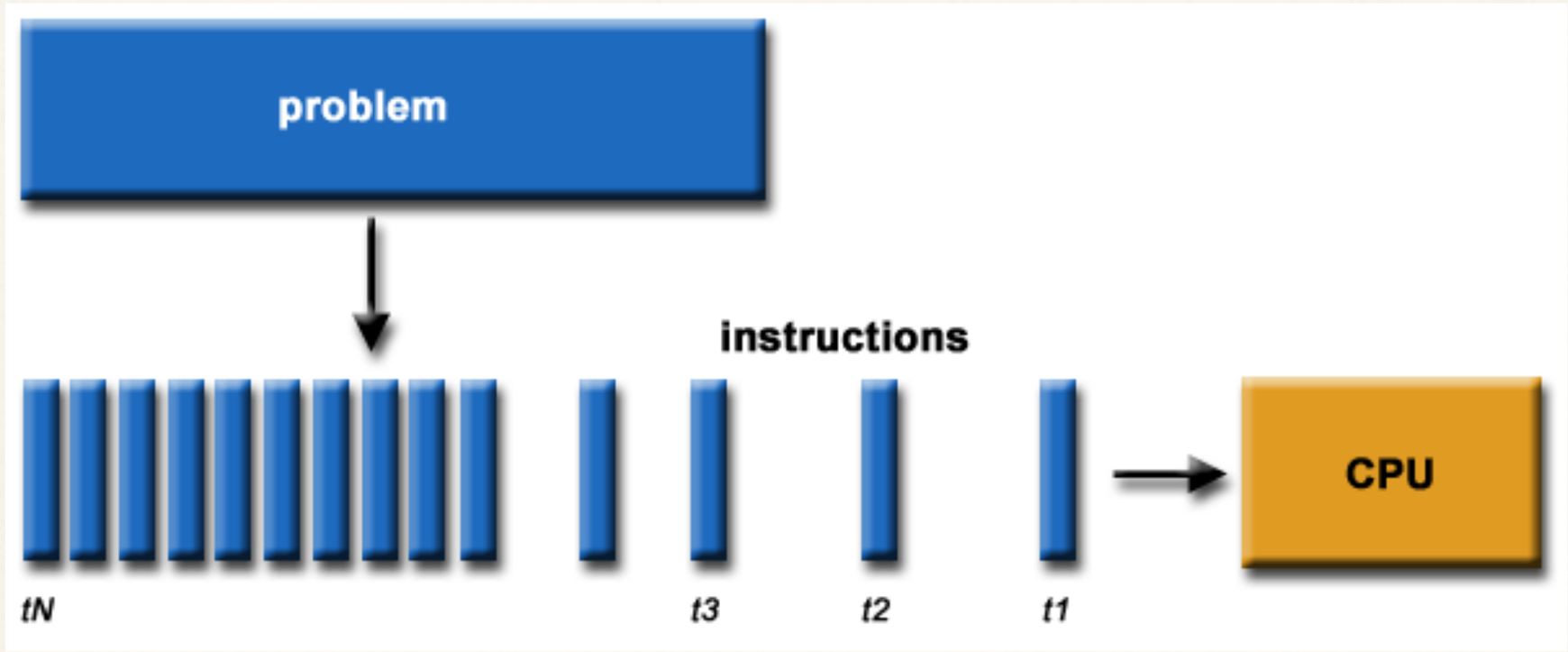
Параллельное программирование многопоточных систем

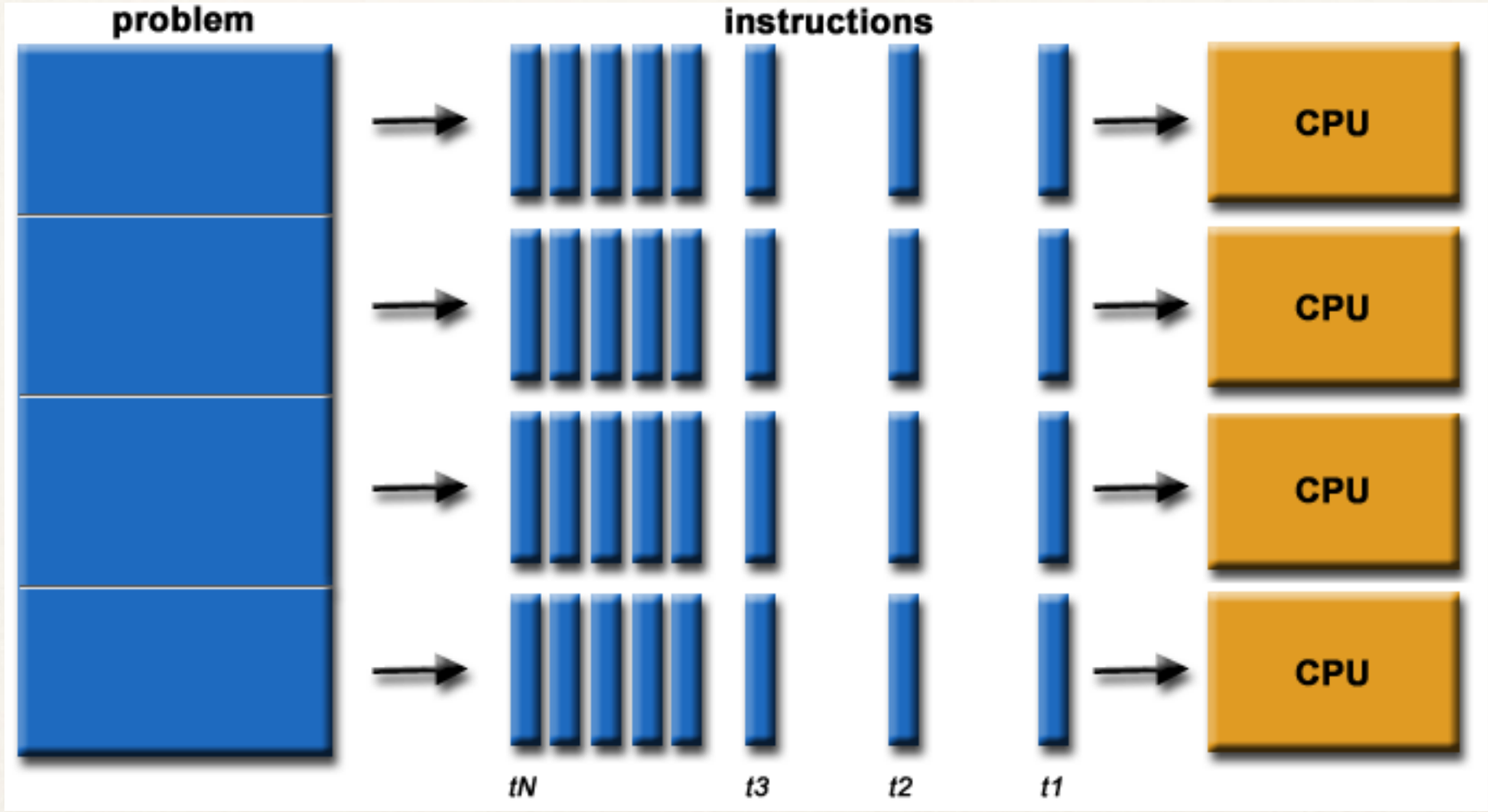
Анна Субботина

14 Сентября 2015

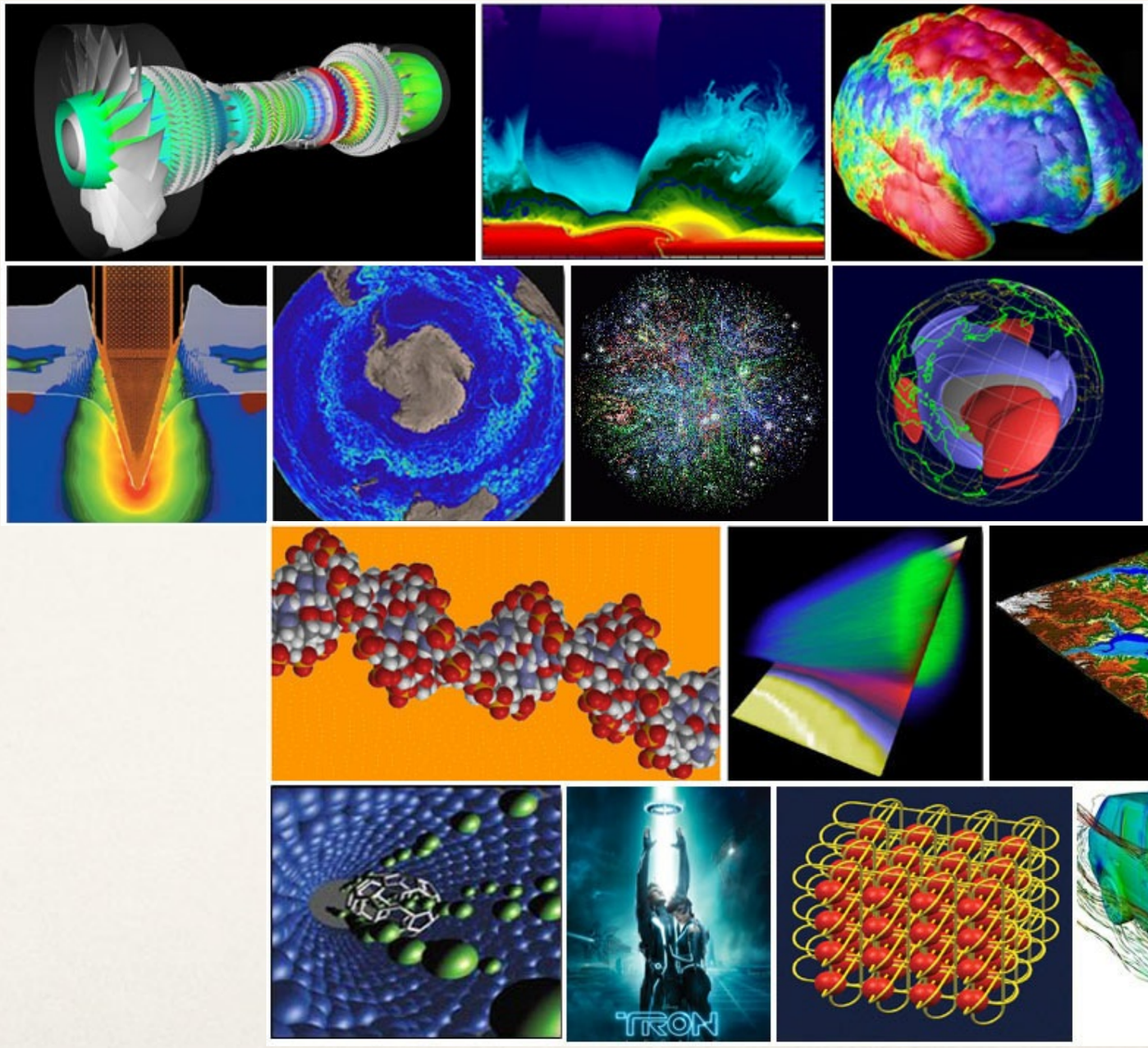
Последовательное исполнение



Параллельное исполнение



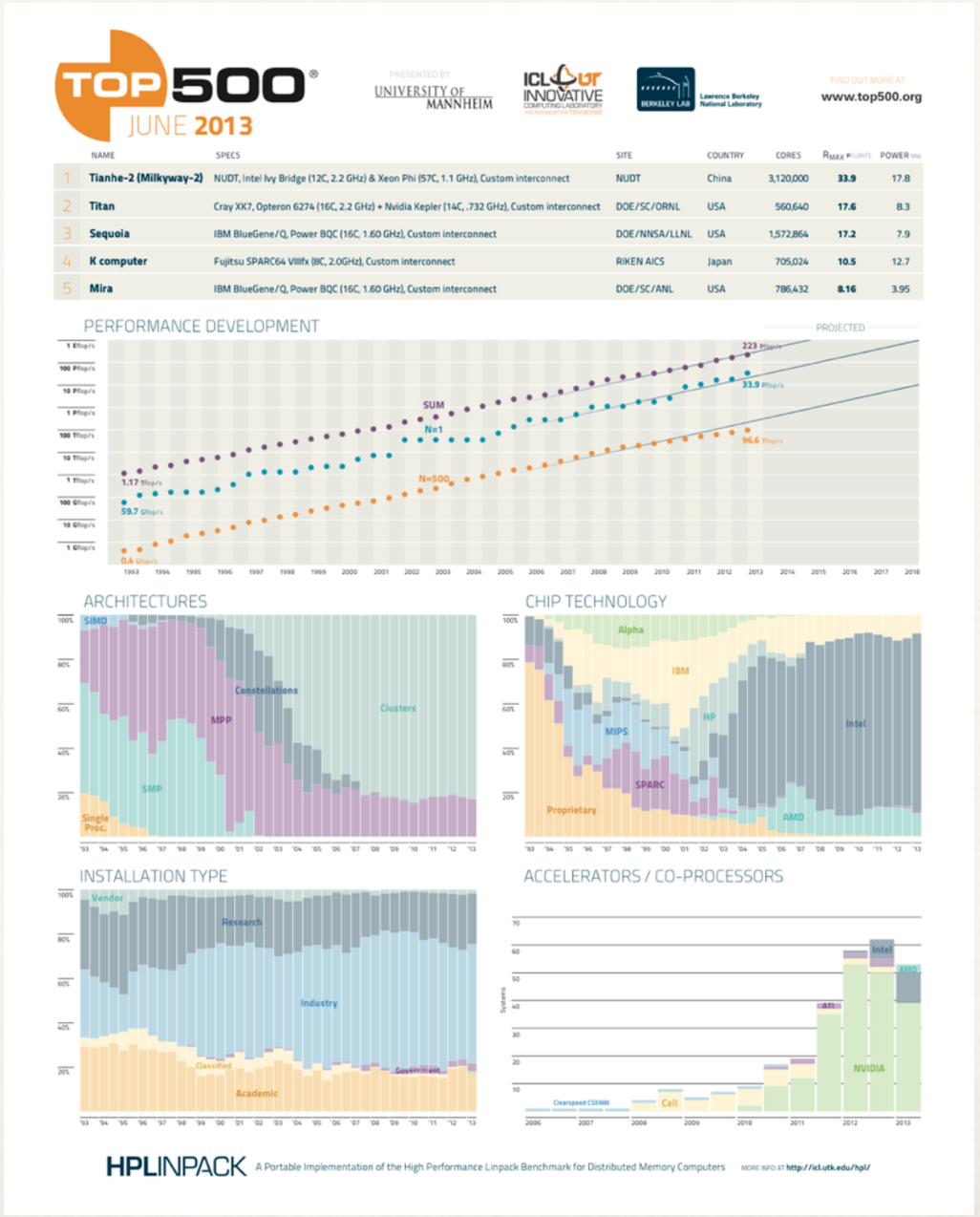
Задачи параллельных вычислений



Преимущества

- Уменьшение времени работы
- Решение очень больших задач
- Одновременное решение нескольких задач
- Использование удаленных распределенных ресурсов
- Обход ограничений последовательных вычислительных систем

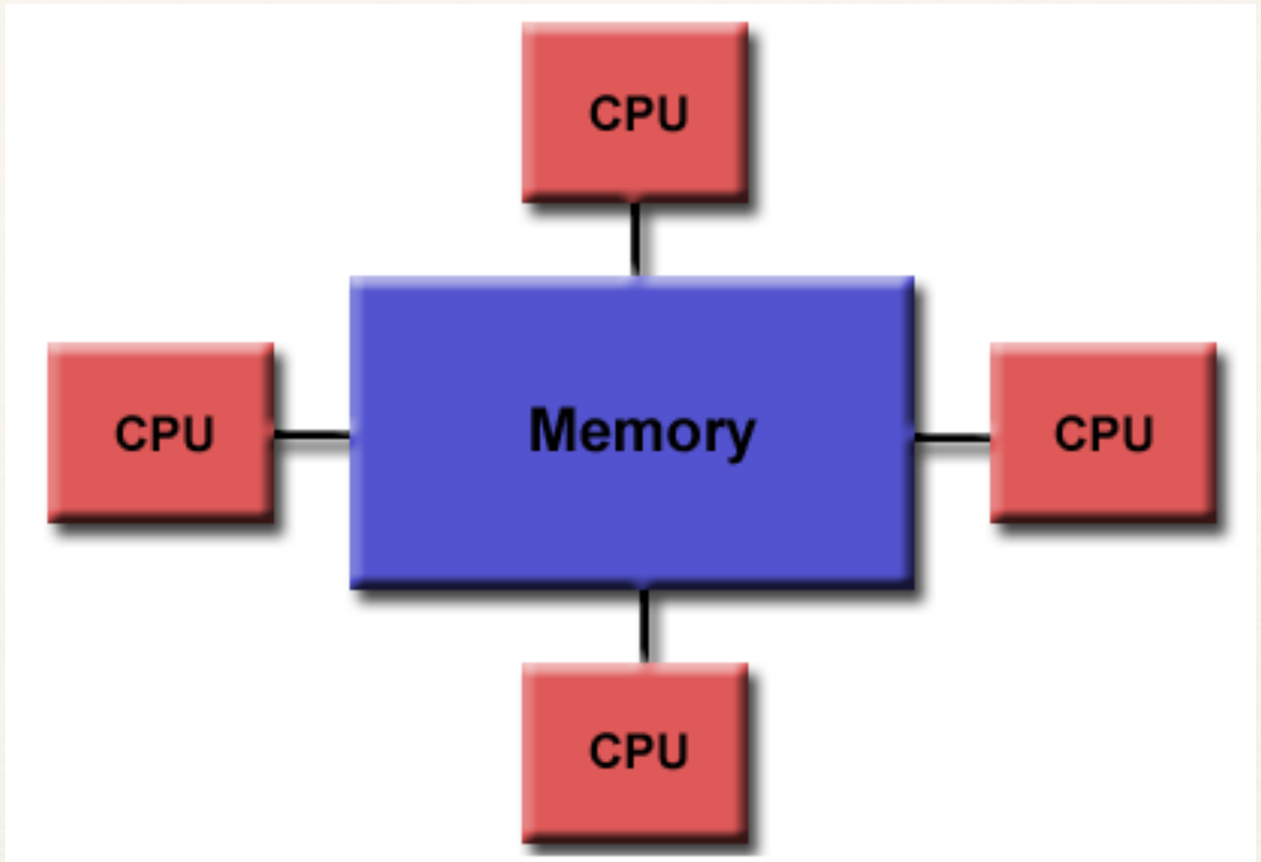
Top 500


TOP500
JUNE 2013
PRESENTED BY
UNIVERSITY OF MANNHEIM
ICL UT INNOVATIVE COMPUTING LABORATORY
BERKELEY LAB Lawrence Berkeley National Laboratory
FIND OUT MORE AT
www.top500.org
NAME SPECS SITE COUNTRY CORES RMAX PFLOP/S POWER MW
1 Tianhe-2 (Milkyway-2) NUDT, Intel Ivy Bridge (12C, 2.2 GHz) & Xeon Phi (57C, 1.1 GHz), Custom interconnect NUDT China 3,120,000 33.9 17.8
2 Titan Cray XK7, Opteron 6274 (16C, 2.2 GHz) + Nvidia Kepler (14C, .732 GHz), Custom interconnect DOE/SC/ORNL USA 560,640 17.6 8.3
3 Sequoia IBM BlueGene/Q, Power BQC (16C, 1.60 GHz), Custom interconnect DOE/NNSA/LLNL USA 1,572,864 17.2 7.9
4 K computer Fujitsu SPARC64 VIIIfx (8C, 2.0GHz), Custom interconnect RIKEN AICS Japan 705,024 10.5 12.7
5 Mira IBM BlueGene/Q, Power BQC (16C, 1.60 GHz), Custom interconnect DOE/SC/ANL USA 786,432 8.16 3.95
PERFORMANCE DEVELOPMENT
PROJECTED
SUM
N=1
N=500
1.17 Pflop/s
59.7 Gflop/s
0.4 Gflop/s
223 Pflop/s
33.9 Pflop/s
96.6 Tflop/s
ARCHITECTURES
SIMD
Constellations
MPP
Clusters
SMP
Single Proc.
CHIP TECHNOLOGY
Alpha
IBM
MIPS
HP
Intel
SPARC
Proprietary
AMD
INSTALLATION TYPE
Vendor
Research
Industry
Classified
Academic
ACCELERATORS / CO-PROCESSORS
Systems
Intel
AMD
ATI
NVIDIA
Clearspeed CSX600
Cell
HPLINPACK A Portable Implementation of the High Performance Linpack Benchmark for Distributed Memory Computers
MORE INFO AT http://icl.utk.edu/hpl/

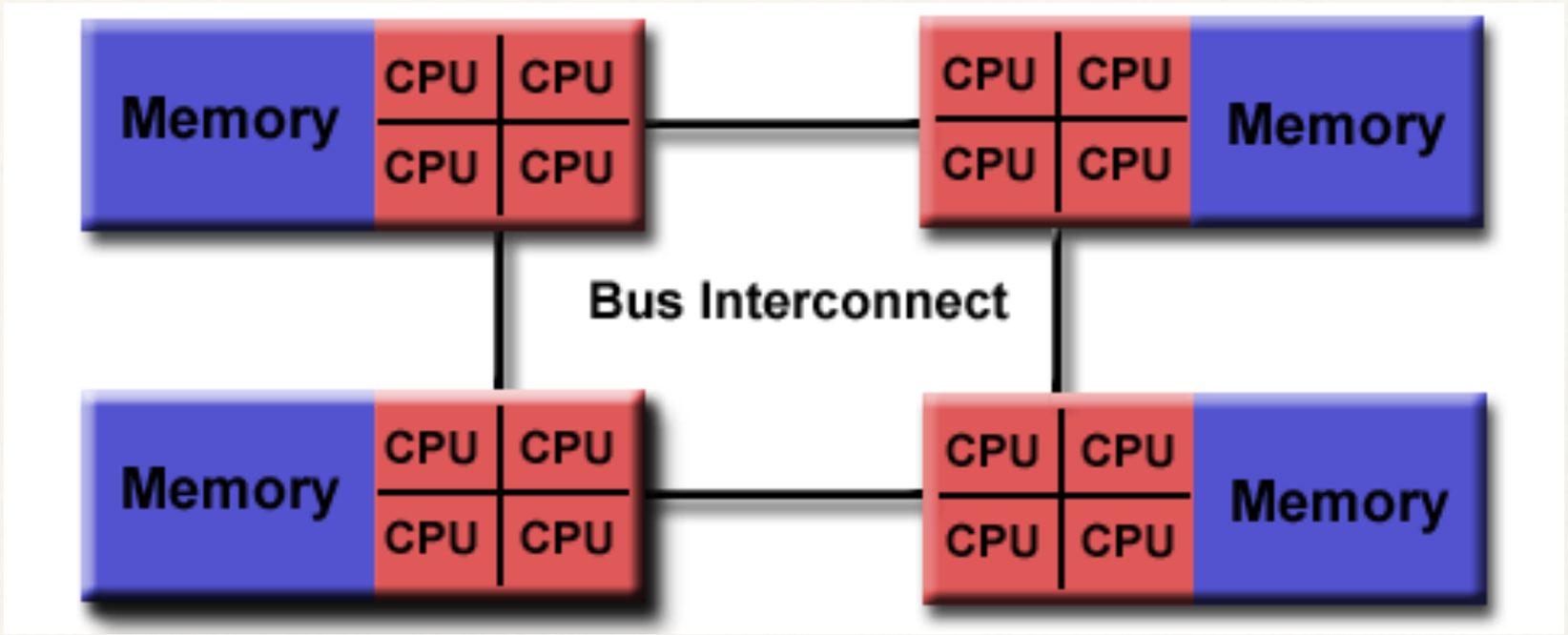
Параллельные архитектуры

- Архитектуры с общей памятью (UMA / NUMA)
- Архитектуры с распределенной памятью
- Гибридные архитектуры

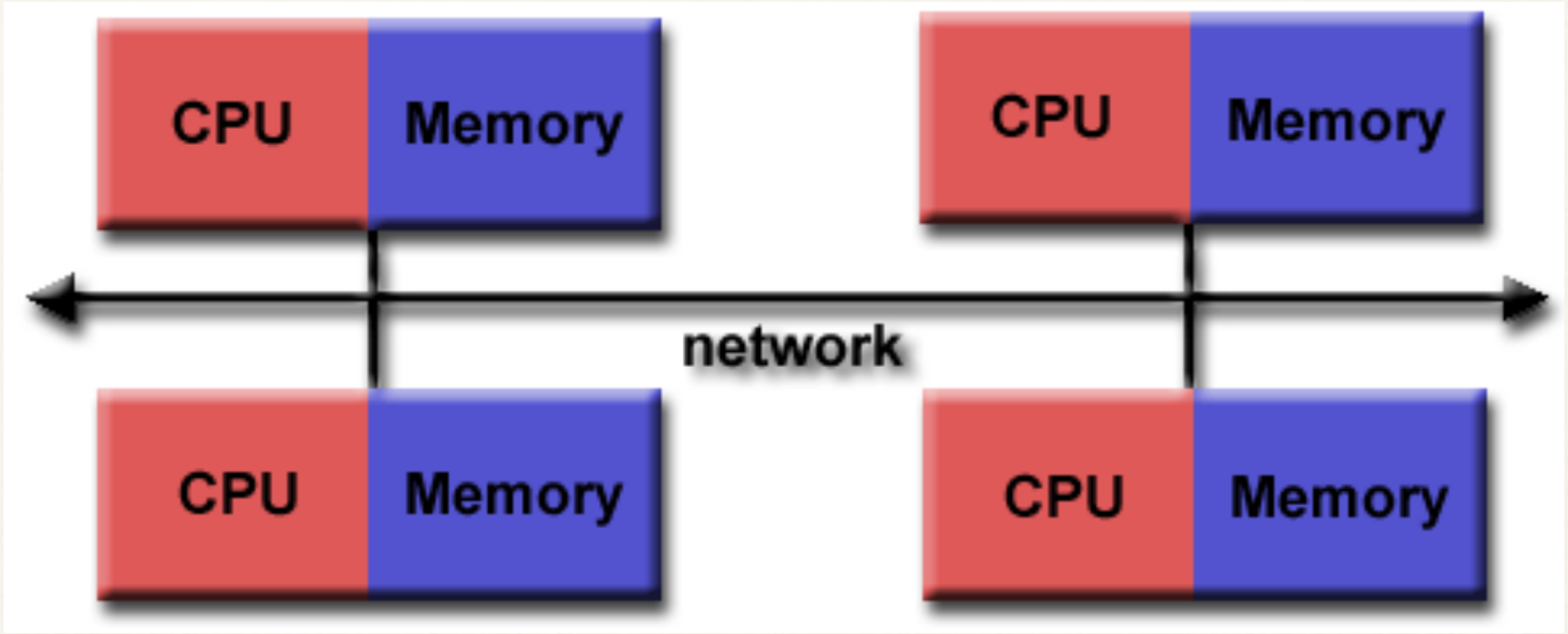
Uniform memory access



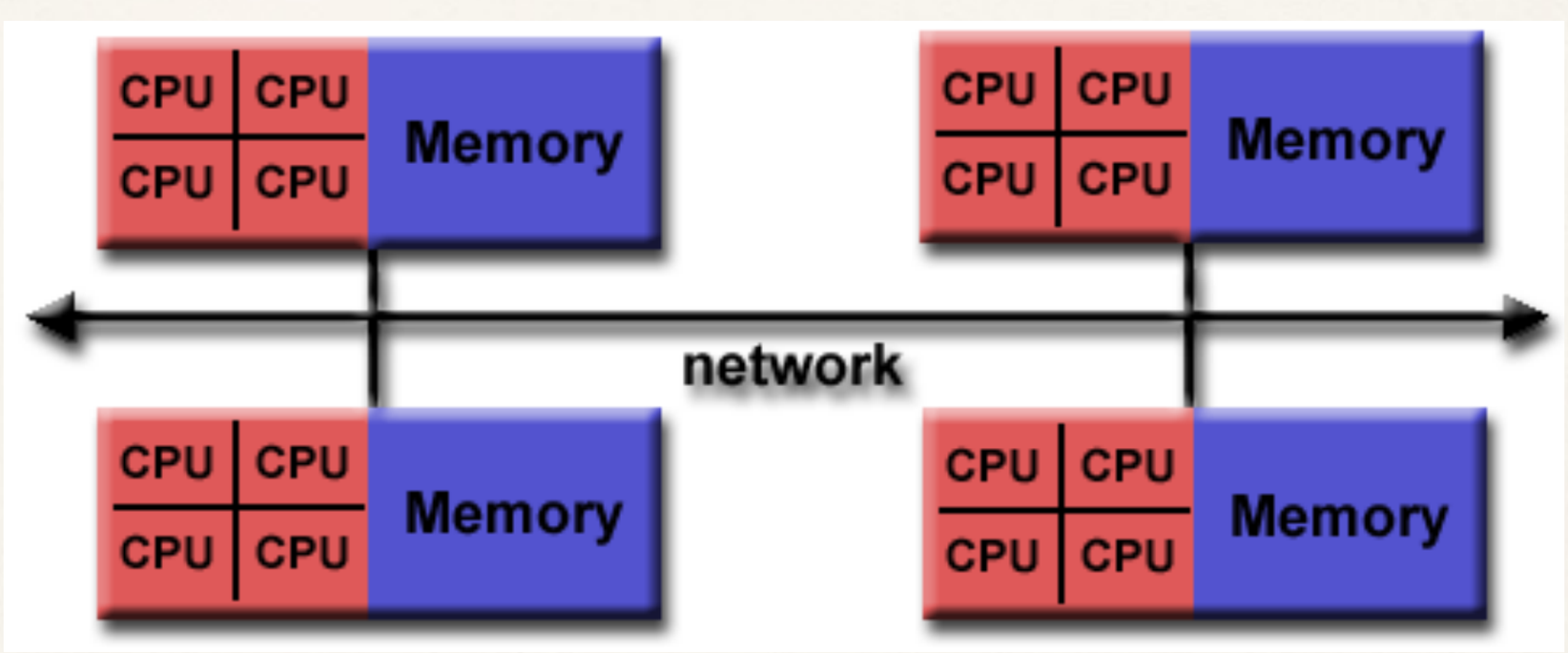
Non-uniform memory access



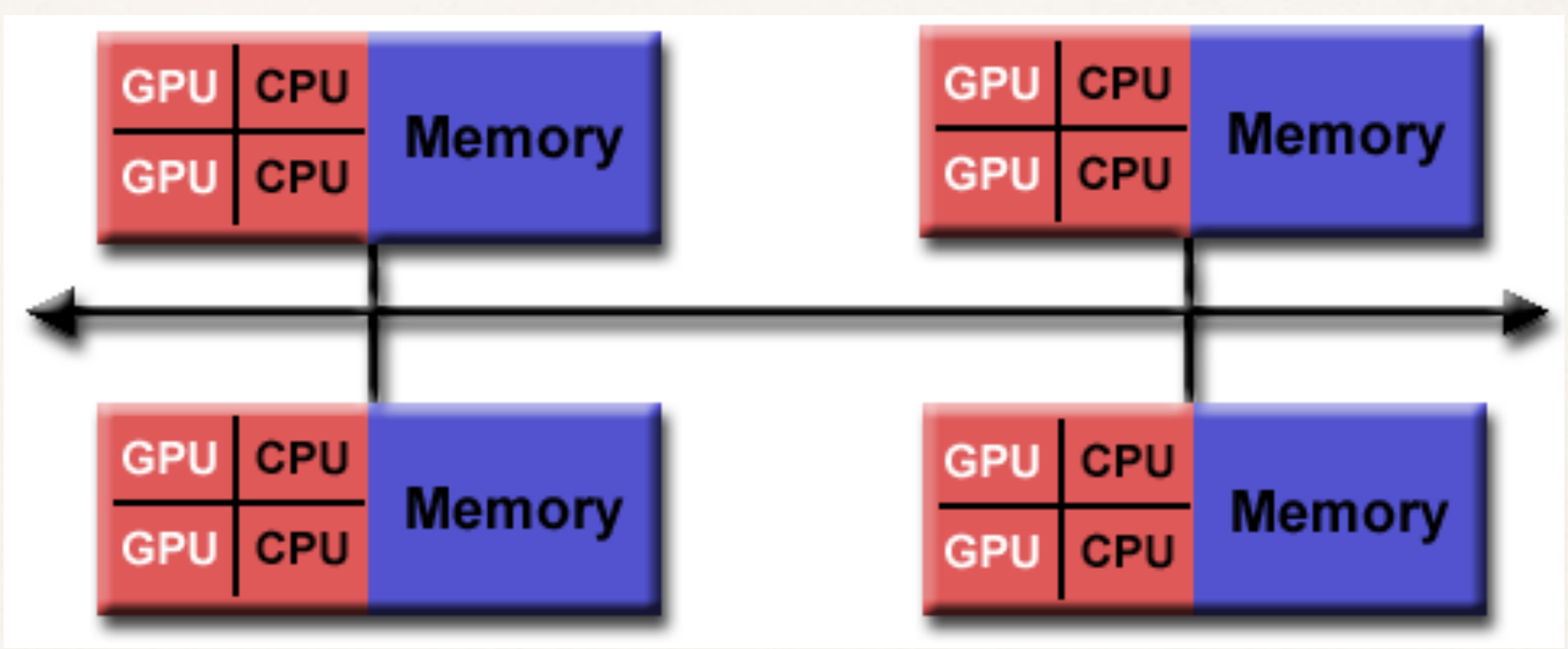
Распределенная память



Гибридные архитектуры



Гибридные архитектуры



Модели параллельных вычислений

- Последовательная модель
- Модель передачи сообщений
- Модель разделенных данных
- Модель разделяемой памяти
- Гибридные

Последовательная модель

- Последовательная программа для автоматического распараллеливания компилятором или специальными программными средствами
  - Преимущество – знакомая парадигма программирования
  - Недостаток – ограниченные возможности автоматического распараллеливания
  - Пример: Intel Parallel Studio

Модель передачи сообщений

- Работающее приложение состоит из набора процессов с различными адресными пространствами. Процессы обмениваются сообщениями с помощью явных send-receive операций
  - Преимущество – полный контроль над выполнением
  - Недостаток – сложность и кропотливость программирования
  - Пример: MPI

Модель разделяемой памяти

- Приложение состоит из набора thread'ов, использующих разделяемые переменные и примитивы синхронизации
- Явные нити исполнения: программирование с использованием библиотечных или системных вызовов для thread'ов
  - Преимущество – переносимость
  - Недостаток – сложность программирования
  - Пример: средства System V IPC

Модель разделяемой памяти

- Приложение состоит из набора thread'ов, использующих разделяемые переменные и примитивы синхронизации
- Программирование на языке высокого уровня с применением прагм
  - Преимущество – легкость программирования
  - Недостаток – сложность контроля над выполнением
  - Пример: OpenMP

Модель разделенных данных

- Приложение состоит из набора процессов, каждый работает со своим набором данных, обмена информацией при работе нет
- Преимущество – легкость реализации
- Недостаток – производительность зависит от конкретной реализации, малый круг задач
- Пример: задачи дешифрования и рендеринга

Создание потоков

- POSIX Threads
- OpenMP
- Intel Threading Building Blocks

POSIX Threads

- Стандарт POSIX реализации потоков выполнения определяет API для управления потоками, их синхронизации и планирования

- Реализации данного API существуют для большого числа UNIX-подобных ОС, а также для Microsoft Windows

POSIX Threads

```
#include <pthread.h>
// …

void *pt_func(void *num) {
        pthread_t thrd_id = pthread_self();
        long thrd_num = (long)num;
        pthread_exit(NULL);
}

int main(int argc, char **argv) {
        int sum = 0, y = 0, i = 0;
        pthread_t pthreads[3];
        for (i = 0; i < 3; i++)
                if (pthread_create(&pthreads[i], NULL, pt_func, (void *)i) > 0) {
                        perror(argv[0]); return errno;
                }
        for (i = 0; i < 3; i++)
                if (pthread_join(pthread[i], NULL)) {
                        perror(argv[0]); return errno;
                }
        return 0;
}
```

POSIX Threads

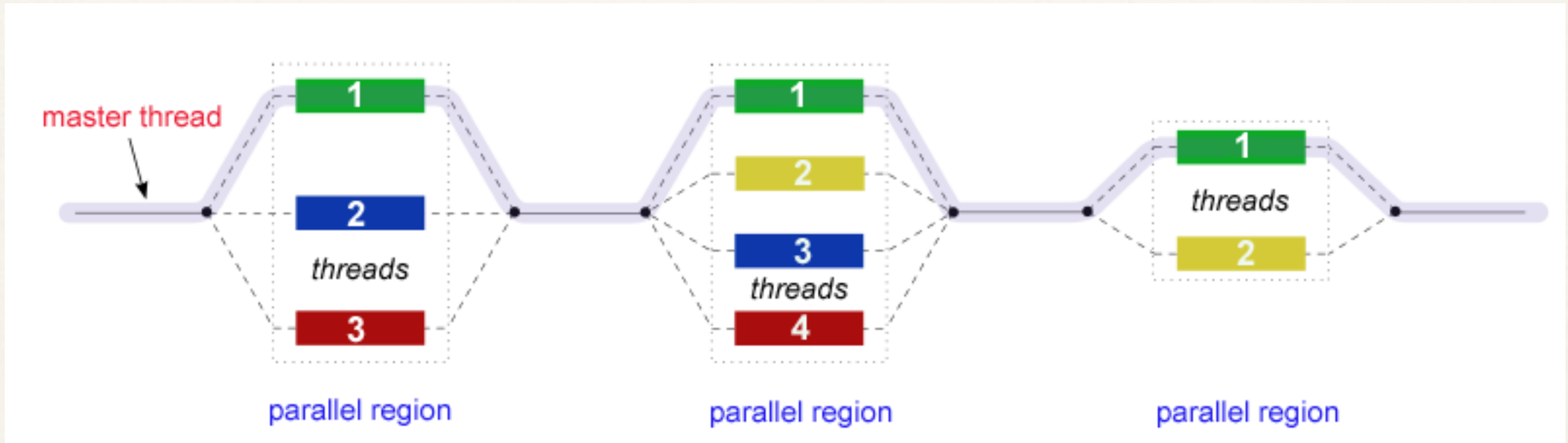
- Компиляция:

gcc pthreads.c -o pthrs -lpthread

OpenMP

- Открытый стандарт (API) для распараллеливания многопоточных приложений на многопроцессорных системах с общей памятью
- Состоит из трех основных компонент API:
  - директив компилятора
  - библиотечных функций / процедур
  - переменных окружения

OpenMP



Синтаксис: #pragma omp directive [clause ...]

OpenMP

```
#include <omp.h>
// …

int main() {
        int sum = 0, val = 0;

#pragma omp parallel shared (sum), private (val)
{
        val = omp_get_thread_num();
        printf("val = %d\n", val);
        sum += val;
}
        printf("sum = %d\n", sum);
        return 0;
}
```

POSIX Threads

- Компиляция:

gcc -fopenmp omp.c -o omp

Intel Threading Building Blocks

- кроссплатформенная библиотека шаблонов C++, разработанная компанией Intel для параллельного программирования
- содержит алгоритмы и структуры данных, позволяющие программисту избежать многих сложностей, возникающих при использовании традиционных реализаций потоков
- все операции трактуются как «задачи», которые динамически распределяются между ядрами процессора
- достигается эффективное использование кэша
- этот подход позволяет программировать параллельные

Intel Threading Building Blocks

- параллельные алгоритмы: for, reduce, do, scan, while, pipeline, sort
- потокобезопасные контейнеры: вектор, очередь, хеш-таблица
- масштабируемые распределители памяти
- мьютексы, атомарные операции
- глобальная временная метка
- планировщик задач
- вычислительный граф

Intel Threading Building Blocks

```
#include <tbb/tbb.h>
#include <tbb/parallel_for.h>
using namespace std;
using namespace tbb;
const int SIZE = 10000000;
class CalculationTask {        // класс-обработчик
        double *myArray;
public:
        // Оператор () выполняется над диапазоном из пространства итераций
        void operator()(const tbb::blocked_range<int> &r) const {
                for (int i = r.begin(); i != r.end(); i++)
                        Calculate(myArray[i]);
        };
        CalculationTask (double *a) : myArray(a) {};
};
int main() {
        double *myArray = new double[SIZE];
        // Запуск параллельного алгоритма for
        tbb::parallel_for(tbb::blocked_range<int>(0, SIZE), CalculationTask(myArray));
        delete[] myArray;
        return 0;
}
```

Intel Threading Building Blocks

- Компиляция:

g++ tbb.cpp -o tbb -ltbb

Расчет времени исполнения кода

- Read Time Stamp Counter (Time Stamp Counter)
- omp_get_wtime()
- clock()
- time()

Read Time Stamp Counter

- ассемблерная инструкция читающая счётчик TSC и возвращающая в регистрах EDX:EAX 64-битное количество тактов с момента последнего сброса процессора
- rdtsc поддерживается в процессорах Pentium и более новых
- имеет точность почти до тика
- скорость, обычно в десятки-сотни тысяч раз быстрее, что очень важно, например, при измерении времени исполнения интенсивных операций, таких, как вызовы функций

Read Time Stamp Counter

```
#if defined(__i386__)
static __inline__ unsigned long long rdtsc(void) {
        unsigned long long int x;
        __asm__ volatile (".byte 0x0f, 0x31" : "=A" (x));
        return x;
}
#elif defined(__x86_64__)
static __inline__ unsigned long long rdtsc(void) {
        unsigned hi, lo;
        __asm__ __volatile__ ("rdtsc" : "=a"(lo), "=d"(hi));
        return ( (unsigned long long)lo)|( ((unsigned long long)hi)<<32 );
}
#endif
int main() {
        long long rdtscStart;
        int i = 0;
        for (i = 0; i < 20; i++)  {
                rdtscStart = rdtsc();
                for (i = 0; i < 10000000; i++) {}
                printf("%lld\r\n", rdtsc() - rdtscStart);
        }
}
```

Расчет времени исполнения кода

```
int main() {
	long long rdtscStart = rdtsc(), ticksDelta;
	long clockStart = clock(), clockDelta; // time_t
	double ompStart = omp_get_wtime();
	long timeStart = time(NULL);

	printf("RDTSC START = %lld\r\n", rdtscStart);
	printf("CLOCK START = %ld\r\n", clockStart);
	printf("OMP   START = %lf\r\n", ompStart);
	printf("TIME  START = %ld\r\n", timeStart);

	for (; i < 1000000000; i++) {}

	printf("RDTSC delta = %lld ticks\r\n", ticksDelta = rdtsc() - rdtscStart);
	printf("CLOCK delta = %ld ms?\r\n", clockDelta = clock() - clockStart);
	printf("OMP   delta = %lf s?\r\n", omp_get_wtime() - ompStart);
	printf("TIME  delta = %ld\r\n", time(NULL) - timeStart);
	printf("RATIO tics/sec %lf\r\n", ticksDelta / clockDelta / 1000.0 );
	return 0;
}
```

Вопросы?