Многопоточное программирование

OpenMP

Open Multi-Proccesing

- Открытый стандарт для распараллеливания программ
- C, C++, Fortran
- Набор директив компилятора, библиотечных процедур и переменных окружения
- http://www.openmp.org/

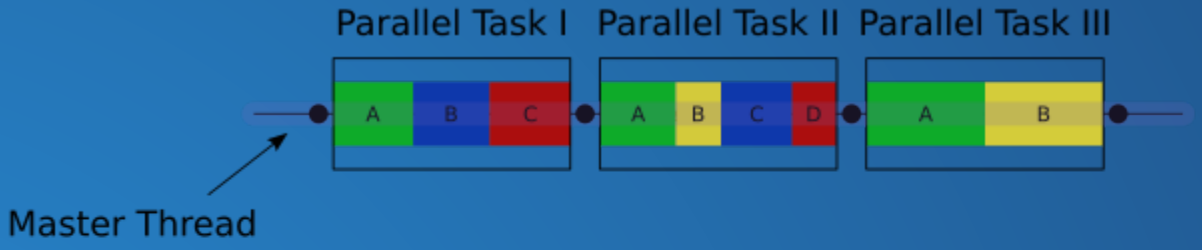
История

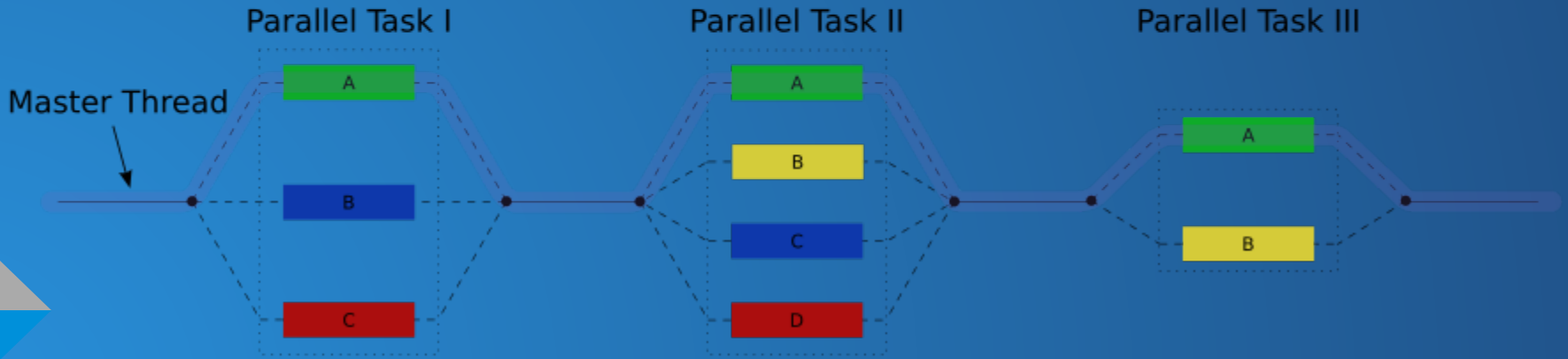
- OpenMP Architecture Review Board (ARB)
- OpenMP for Fortran 1.0, 1997
- OpenMP 2.0 2002 поддержка C++
- OpenMP 2.5 2005
- OpenMP 3.0 2008
- OpenMP 3.1 2011
- OpenMP 4.0 2013

Поддержка

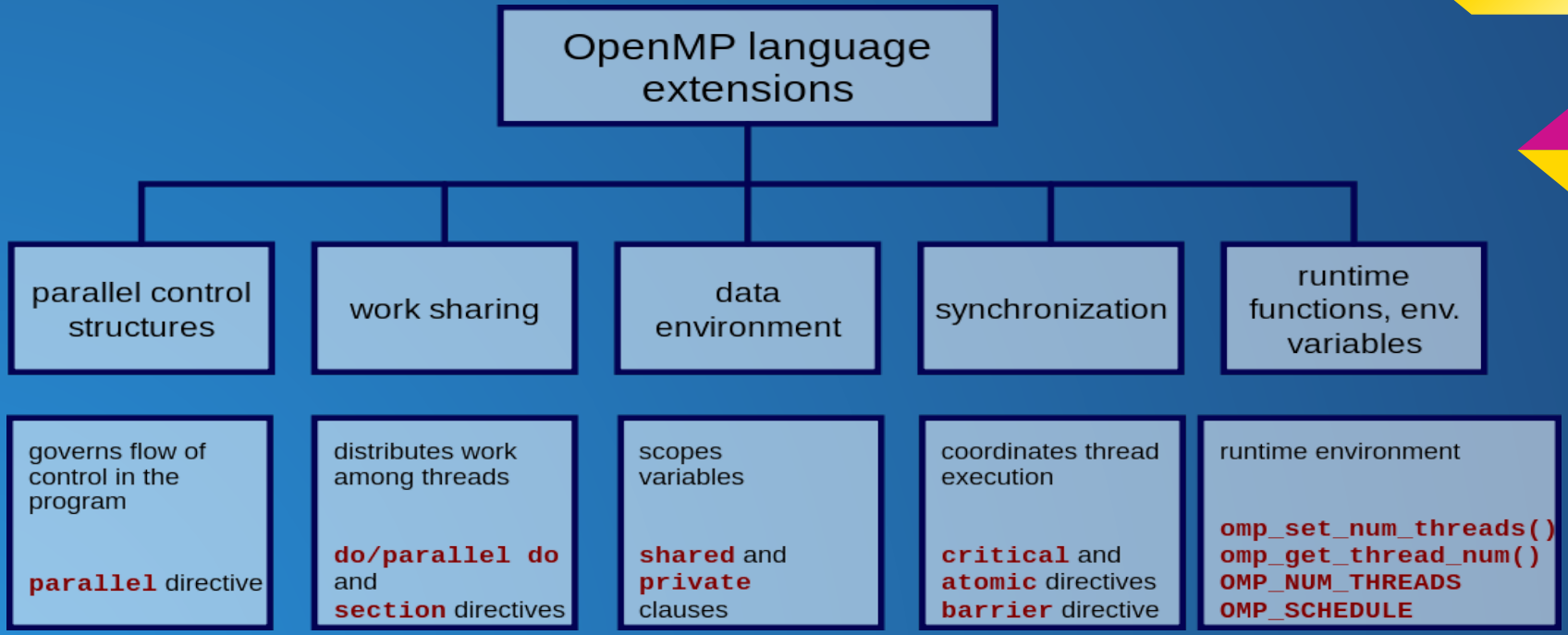
- OpenMP 2.5
  - GCC 4.2 и выше
  - Microsoft Visual C++ 2008 и выше
  - Intel C/C++ Compiler 10.1 и выше
- OpenMP 3.0
  - GCC 4.4 и выше
  - Intel C/C++ Compiler 11.0 и выше

Fork-Join model

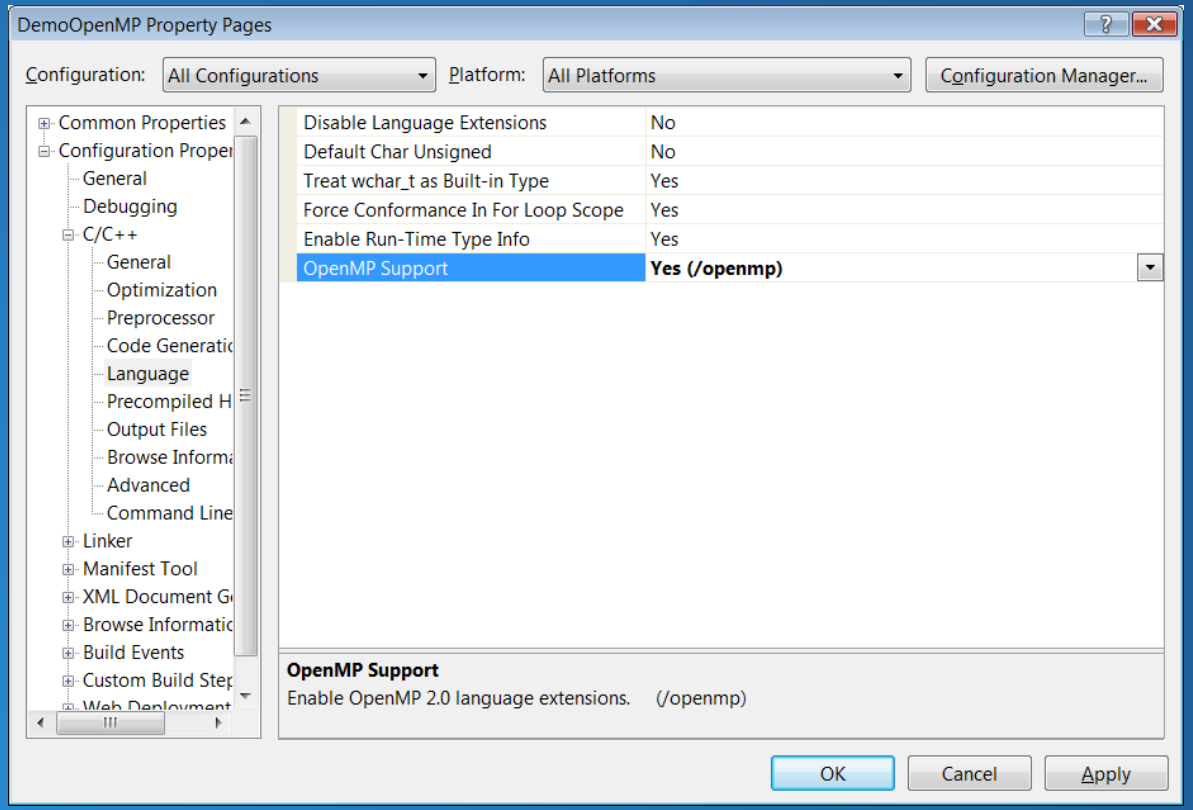




Ядро


OpenMP language extensions
parallel control structures
work sharing
data environment
synchronization
runtime functions, env. variables
governs flow of control in the program
parallel directive
distributes work among threads
do/parallel do and
section directives
scopes variables
shared and
private
clauses
coordinates thread execution
critical and
atomic directives
barrier directive
runtime environment
omp_set_num_threads()
omp_get_thread_num()
OMP_NUM_THREADS
OMP_SCHEDULE

OpenMP & Visual Studio



OpenMP

- #include <omp.h>
- Секции которые хочется распаралелить помечаются специальными директивами
- Каждоы поток получает уникальны id
  - 0 - master
  - omp_get_thread_num()

Директива pragma

```
#pragma omp <директива> [раздел [ [,] раздел]...]
```

Директива parallel

```
#pragma omp parallel [раздел[ [,] раздел]...]
структурированный блок
```

Hello World

```
#include "stdafx.h"
#include <omp.h>


int _tmain(int argc, _TCHAR* argv[])
{
    #pragma omp parallel
        printf("Hello world\n");
    return 0;
}
```

Hello World 2

```
int _tmain(int argc, _TCHAR* argv[])
{
    int thread_id, thread_num;
    #pragma omp parallel private(thread_id)
    {
        thread_id = omp_get_thread_num();
        printf("%d Hello world\n", thread_id);

        if (thread_id == 0){
            thread_num = omp_get_num_threads();
            printf("Numeber of threads %d\n", thread_num);
        }
    }

    return 0;
}
```

Атрибуты видимости данных

- private (list)
- firstprivate (list)
- lastprivate (list)
- threadprivate (list)
- shared (list)
- default (shared | none)

Директива for

Разделение итераций по разным потокам с учетом дополнительных налагаемых условий.

```
int _tmain(int argc, _TCHAR* argv[])
{
    #pragma omp parallel
    #pragma omp for
    for (int i = 0; i < 10; i++){
        printf("%d: Hello world from thread %d\n", i, omp_get_thread_num());
    }

    return 0;
}
```

Пример. Интеграл.

```
int _tmain(int argc, _TCHAR* argv[])
{
    time_t start_time;
    time_t finish_time;

    int eteration_count = 10000000000000000; // количество итераций
    int a = 0;                          // левая граница интегрирования
    int b = 1;                          // правая граница интегрирования
    int x;

    time(&start_time);

    double step = (b - a) / eteration_count;
    double total = (liner(a) + liner(b))*step / 2;
    for (int i = 0; i < eteration_count; ++i)
    {
        x = a + i* step;
        total += liner(x)*step;
    }
    time(&finish_time);

    std::cout << "Result is: " << total << " Operation time: " << difftime(finish_time, start_time) << std::endl
    return 0;
}
```

Пример. Интеграл.

```
int _tmain(int argc, _TCHAR* argv[])
{
    time_t start_time;
    time_t finish_time;

    int eteration_count = 10000000000000000; // количество итераций
    int a = 0;                          // левая граница интегрирования
    int b = 1;                          // правая граница интегрирования

    time(&start_time);

    double total = 0;
    double x;
    double step = (float)(b - a) / eteration_count;

    total = (liner(a) + liner(b))*step / 2;
    #pragma omp parallel

    for (int i = 0; i < eteration_count; ++i){
        x = a + i*step;
        total += liner(x)*step;
    }

    time(&finish_time);
    std::cout << "Result is: " << total << " Operation time: " << difftime(finish_time, start_time) << std::endl;
    return 0;
}
```

Пример

```
int _tmain(int argc, _TCHAR* argv[])
{
    time_t start_time;
    time_t finish_time;

    int eteration_count = 10000000000000000; // количество итераций
    int a = 0;                          // левая граница интегрирования
    int b = 1;                          // правая граница интегрирования

    time(&start_time);

    double total = 0;
    double x;
    double step = (float)(b - a) / eteration_count;

    total = (liner(a) + liner(b))*step / 2;
    #pragma omp parallel
    #pragma omp parallel for private(x)
    for (int i = 0; i < eteration_count; ++i){
        x = a + i*step;
        total += liner(x)*step;
    }

    time(&finish_time);
    std::cout << "Result is: " << total << " Operation time: " << difftime(finish_time, start_time) << std::endl;
    return 0;
}
```

Пример. Инеграл

```
int _tmain(int argc, _TCHAR* argv[])
{
    time_t start_time;
    time_t finish_time;

    int eteration_count = 10000000000000000; // количество итераций
    int a = 0;                          // левая граница интегрирования
    int b = 1;                          // правая граница интегрирования

    time(&start_time);

    double total = 0;
    double x;
    double step = (float)(b - a) / eteration_count;

    total = (liner(a) + liner(b))*step / 2;
    #pragma omp parallel
    #pragma omp parallel for private(x)
    for (int i = 0; i < eteration_count; ++i){
        x = a + i*step;
        total += liner(x)*step;
    }

    time(&finish_time);
    std::cout << "Result is: " << total << " Operation time: " << difftime(finish_time, start_time) << std::endl;
    return 0;
}
```

Пример

```
int _tmain(int argc, _TCHAR* argv[])
{
    time_t start_time;
    time_t finish_time;

    int eteration_count = 10000000000000000; // количество итераций
    int a = 0;                       // левая граница интегрирования
    int b = 1;                       // правая граница интегрирования

    time(&start_time);

    double total = 0;
    double x;
    double step = (float)(b - a) / eteration_count;

    int num_of_threads = omp_get_num_threads();
    double* totals = new double[num_of_threads];
    total = (liner(a) + liner(b))*step / 2;
    #pragma omp parallel
    {

        int id = omp_get_thread_num();
        #pragma omp for private(x) nowait
        for (int i = 0; i < eteration_count; ++i){
            x = a + i*step;
            totals[id] += liner(x)*step;
        }
        #pragma omp critical
        total += totals[id];
    }

    time(&finish_time);
    std::cout << "Result is: " << total << " Operation time: " << difftime(finish_time, start_time) << std::endl;


    delete totals;
    return 0;
}
```

Условия синхронизации

- *critical*
- *atomic*
- *ordered*
- *barrier*
- *nowait*

Пример. Интеграл

```
int _tmain(int argc, _TCHAR* argv[])
{
    time_t start_time;
    time_t finish_time;

    int eteration_count = 10000000000000000; // количество итераций
    int a = 0;                               // левая граница интегрирования
    int b = 1;                               // правая граница интегрирования

    time(&start_time);

    double total = 0;
    double x;
    double step = (float)(b - a) / eteration_count;

    total = (liner(a) + liner(b))*step / 2;
    #pragma omp parallel
    #pragma omp for private(x) reduction(+:total)
    for (int i = 0; i < eteration_count; ++i){
        x = a + i*step;
        total += liner(x)*step;
    }

    time(&finish_time);
    std::cout << "Result is: " << total << " Operation time: " << difftime(finish_time, start_time) << std::endl;
    return 0;
}
```

Редукция

Operation	Temporary private variable initialization
+ (addition)	0
- (subtraction)	0
* (multiplication)	1
& (bitwise and)	~0
| (bitwise or)	0
^ (bitwise exclusive or)	0
&& (conditional and)	1
|| (conditional or)	0

Оптимизация циклов

Объединений
Развертывание выложенных циклов

Развертывание вложенных циклов

```
for (j = 0; j<N; j++) {
 for (i = 0; i<M; i++) {
        A[i][j] = work(i, j);
    }
}

#pragma omp parallel for private (ij , j, i)
for (ij = 0; ij <N*M; ij++) {
     j = ij / N;
     i = ij%M;
    A[i][j] = work(i, j);

}
```

Пример. Число Фибоначчи

$$F_0 = 0, \qquad F_1 = 1, \qquad F_n = F_{n-1} + F_{n-2}, \quad n \geqslant 2, \quad n \in \mathbb{Z}.$$

```
long long  fibonachi(long int n)
{
    if (n == 0 || n == 1)
        return 1;
    return fibonachi(n - 1) + fibonachi(n - 2);
}
```

Пример. Фибоначчи

```
int _tmain(int argc, _TCHAR* argv[])
{
    long long fib[40];
    time_t begin, end;
    time(&begin);

    #pragma omp parallel
    {
        #pragma omp for
        for (int i = 0; i < 40; i++){
            fib[i] = fibonachi(i);
        }
    }
    time(&end);

    printf("%f\n", difftime(end, begin));

    return 0;
}
```

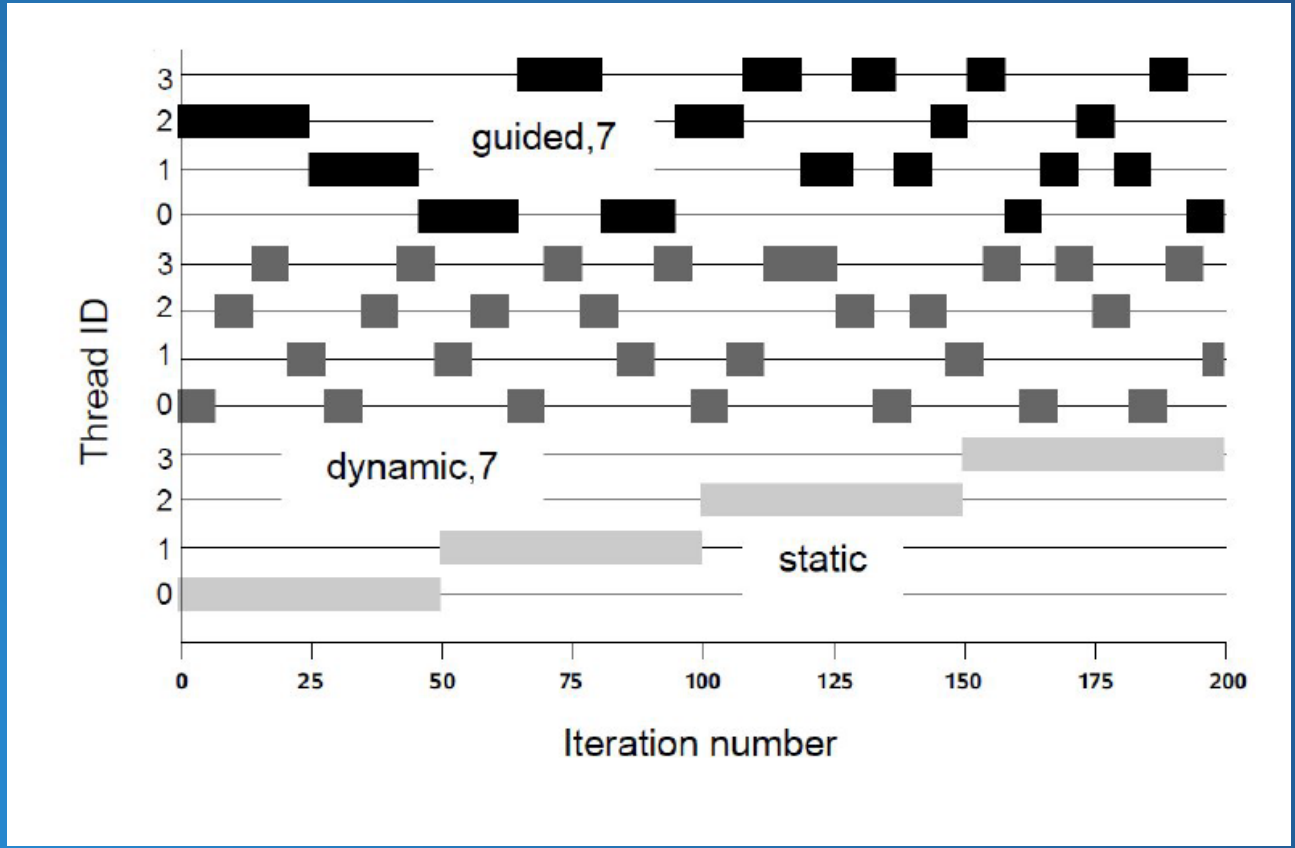
Пример. Фибоначчи

```
int _tmain(int argc, _TCHAR* argv[])
{
    long long fib[40];
    time_t begin, end;
    time(&begin);

    #pragma omp parallel
    {
        #pragma omp for ordered schedule(dynamic)
        for (int i = 0; i < 40; i++){
            fib[i] = fibonachi(i);
        }
    }
    time(&end);

    printf("%f\n", difftime(end, begin));

    return 0;
}
```

Распределение итераций между потоками

- Атрибут Schedule(type, chunk)
- Type
  - static
  - dynamic
  - guided
  - runtime

Распределение этераций между потоками



Пример. Фибоначчи

```
int _tmain(int argc, _TCHAR* argv[])
{
    long long fib[40];
    time_t begin, end;
    time(&begin);

    #pragma omp parallel
    {
        #pragma omp for schedule(static,1)
        for (int i = 0; i < 40; i++){
            fib[i] = fibonachi(i);
            printf("%i ", fib[i]);

        }
    }
    time(&end);

    printf("%f\n", difftime(end, begin));

    return 0;
}
```

Пример. Фибоначчи

```
int _tmain(int argc, _TCHAR* argv[])
{
    long long fib[40];
    time_t begin, end;
    time(&begin);

    #pragma omp parallel
    {
        #pragma omp for ordered schedule(static,1)
        for (int i = 0; i < 40; i++){
            fib[i] = fibonachi(i);
            #pragma omp ordered
            {
                printf("%i ", fib[i]);
            }
        }
    }
    time(&end);

    printf("%f\n", difftime(end, begin));

    return 0;
}
```

Section

```
int _tmain(int argc, _TCHAR* argv[])
{
    #pragma omp parallel sections
    {
        #pragma omp section
        {
            TaskA();
        }
        #pragma omp section
        {
            TaskC();
        }
        #pragma omp section
        {
            TaskC();
        }
    }
    return 0;
}
```

Накладные расходы

Constructs	Cost (in microseconds)	Scalability
parallel	1.5	Linear
Barrier	1.0	Linear or O(log(n))
schedule(static)	1.0	Linear
schedule(guided)	6.0	Depends on contention
schedule(dynamic)	50	Depends on contention
ordered	0.5	Depends on contention
Single	1.0	Depends on contention
Reduction	2.5	Linear or O(log(n))
Atomic	0.5	Depends on data-type and hardware
Critical	0.5	Depends on contention
Lock/Unlock	0.5	Depends on contention

Преимущества и недостатки